Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження можливостей .NET MAUI в обробці данних
в обробці данних картографічними API для навігаційних додатків

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
(підпис) Глива В.А. (прізвище та ініціали)

Керівник (підпис) Мудрик І.Я. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)
Рецензент (підпис) Яцишин В.В. (прізвище та ініціали)

Тернопіль 2024

АНОТАЦІЯ

Магістерська робота на тему «Дослідження можливостей .NET MAUI в обробці даних картографічними API для навігаційних додатків» Василь Глива. – Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група Спм-61 Тернопіль, 2024.

Метою магістерської роботи є дослідження можливостей використання платформи .NET MAUI для обробки даних картографічними API з метою створення сучасних навігаційних додатків. Розроблений додаток дозволяє будувати маршрути, взаємодіяти з картами в реальному часі, зберігати точки інтересу користувача, а також забезпечує інтеграцію з популярними картографічними сервісами, такими як Google Maps API.

Під час виконання роботи було використано платформу .NET MAUI для створення кросплатформного інтерфейсу, технологію ASP.NET Core для розробки серверної частини, а також SQLite для локального зберігання даних. Середовище розробки Visual Studio надало можливість реалізувати всі переваги сучасної інтеграції з API.

У ході роботи було проаналізовано існуючі картографічні сервіси та їх SDK, визначено їх переваги й недоліки. Результатом стало створення програмного забезпечення, яке забезпечує оптимальну побудову маршрутів та зручне керування географічними даними, що робить його ефективним рішенням для туристичних і транспортних додатків. Проведено проектування бази даних та архітектури програмної системи.

Ключові слова: .NET MAUI, ASP.NET CORE, GOOGLE MAPS API, КАРТОГРАФІЯ, НАВІГАЦІЙНІ ДОДАТКИ, SQLITE, КРОСПЛАТФОРМНИЙ ДОДАТОК.

ABSTRACT

Master's thesis on the topic "Exploring the Capabilities of .NET MAUI in Processing Data with Mapping APIs for Navigation Applications" Vasyl Hlyva. – Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group Spm-61.

The goal of this master's thesis is to explore the potential of .NET MAUI for processing data using mapping APIs in the development of modern navigation applications. The developed application enables route building, real-time map interaction, saving points of interest, and integration with popular mapping services such as Google Maps API.

During the research, the .NET MAUI platform was utilized to create a cross-platform interface, ASP.NET Core was employed for backend development, and SQLite was used for local data storage. JetBrains Rider provided the tools needed to leverage the advantages of modern API integration.

The work included an analysis of existing mapping services and their SDKs, identifying their strengths and weaknesses. The result is a software application that offers optimal route building and convenient management of geographic data, making it an efficient solution for tourism and transportation-related applications. Database design and system architecture were also developed.

Keywords: .NET MAUI, ASP.NET CORE, GOOGLE MAPS API, MAPPING, NAVIGATION APPLICATIONS, SQLITE, CROSS-PLATFORM APPLICATION.

ЗМІСТ

КВАЛІФІКАЦІЙНА РОБОТА....0
ВСТУП....8
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ....10
1 АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ....11
1.1.Постановка задачі....11
1.1.1.Функціональні вимоги....13
1.1.2.Нефункціональні вимоги....14
1.2.Розгляд аналогів ПЗ....16
1.3.Вибір моделі розробки....17
1.4.Вибір технологій та інструментів для розробки....19
1.4.1.Мова програмування C#....19
1.4.2.Фреймворк .Net....21
1.4.3. NET MAUI як технологія для розробки клієнтських додатків....23
1.4.4.MVVM (Model-View-ViewModel) Архітектурний патерн....25
1.4.5.Середовище розробки JetBrains Rider....26
1.4.6.База даних MySql....28
1.4.7.SQLite в .NET MAUI....30
2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ....33
2.1 Проблематика та доцільність розробки....33
2.2 Управління користувачами....36
2.3 Розгортання бекенд-сервісу....37
2.5 Розробка мобільного додатку....38
2.5 Імплементація сервісу для навігації....39
2.6. Обробка данних користувачів....41
2.7 Опис та побудова маршруту....42
Призначення класу:....42
2.8 Алгоритм побудови маршрутів для користувача....43
3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ:....45
3.1 Структура проекту....45
3.2 Тестування бекенд сервісу....47
Для тестування бекенд сервісу використовуватимемо Postman:....47
3.3.Тестування мобільного додатку:....49
4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ....55
4.1 Охорона праці....55
4.2 Дослідження стійкості роботи систем обробки даних в умовах надзвичайних ситуацій....57

ВИСНОВКИ..... 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 61
ДОДАТКИ..... 63

ВСТУП

Експлуатація сучасних навігаційних додатків вимагає глибоких знань у сфері роботи з картографічними даними, геолокаційними сервісами та ефективної взаємодії користувачів із системою. Зважаючи на те, що кількість даних, які потребують обробки (наприклад, маршрути, точки інтересу, історія користувачів), постійно зростає, важливим стає створення програмного забезпечення, яке оптимізує ці процеси.

Інтеграція картографічних API у навігаційні додатки дозволяє вирішити широкий спектр задач: від побудови маршрутів та відображення карт до автоматичного збору та аналізу даних для індивідуального використання. При цьому особливо важливими є простота використання, висока швидкість обробки даних та інтуїтивно зрозумілий інтерфейс.

Сучасні дослідження показують, що з розвитком технологій зростає популярність навігаційних додатків. Це робить актуальним впровадження програм, здатних забезпечити якісний і швидкий доступ до картографічної інформації.

Метою магістерської роботи є дослідження можливостей .NET MAUI в обробці даних картографічними API для розробки сучасних навігаційних додатків.

Основними завданнями дослідження є:

Аналіз предметної області та формулювання проблеми.

Розробка моделі інтеграції картографічних API з навігаційним додатком.

Реалізація програмного забезпечення на основі .NET MAUI з інтеграцією Google Maps API.

Проведення тестування системи для оцінки її продуктивності та зручності використання.

Актуальність теми зумовлена стрімким розвитком технологій у сфері

картографії та навігації, а також зростаючою потребою у створенні зручних і функціональних рішень для користувачів. Наявні системи часто мають обмежену функціональність, застарілі інтерфейси та складність в інтеграції з іншими сервісами. Тому створення сучасного навігаційного додатку на основі .NET MAUI та картографічних API є актуальним і необхідним.

Для реалізації проекту було обрано платформу .NET MAUI завдяки її перевагам у розробці кросплатформних додатків. Для збереження даних використовується база даних SQLite, що забезпечує швидкий доступ до даних. Середовище розробки JetBrains Rider надає потужні інструменти для створення інтерактивного інтерфейсу та роботи з базою даних.

Результати розробки можуть бути використані як у персональних, так і в комерційних цілях, наприклад, для створення туристичних додатків або сервісів доставки.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

.NET MAUI (Multi-platform App UI) – кросплатформна платформа для розробки користувацьких інтерфейсів.

API (Application Programming Interface) – інтерфейс програмування додатків, що забезпечує взаємодію між програмним забезпеченням і зовнішніми службами.

SQLite – вбудована реляційна база даних для локального зберігання даних у додатках.

UI (User Interface) – користувацький інтерфейс.

UX (User Experience) – користувацький досвід.

GPS (Global Positioning System) – глобальна система позиціонування, що визначає місцезнаходження користувача.

ASP.NET Core – кросплатформний фреймворк для створення веб-додатків і API.

SDK (Software Development Kit) – набір інструментів для розробки програмного забезпечення.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту, що використовується для взаємодії з API.

XAML (Extensible Application Markup Language) – мова розмітки для створення інтерфейсів у .NET MAUI.

1 АНАЛІЗ ВИМОГ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1.Постановка задачі

Завданням магістерської роботи є дослідження можливостей використання платформи .NET MAUI для обробки даних за допомогою картографічних API і створення сучасного навігаційного додатку. Розроблена система має забезпечити зручність і ефективність взаємодії користувача з картографічними сервісами, а також оптимізувати обробку географічних даних.

Програмні засоби системи повинні функціонувати стабільно навіть за умови обробки численних одночасних запитів. Система повинна гарантувати безпеку та точність обробки даних, а також відповідати сучасним стандартам зручності й інтерактивності.

Основні завдання, які необхідно вирішити:

- Розробка архітектури системи: спроєктувати багатошарову архітектуру з розподілом відповідальності між клієнтською, серверною частиною та базою даних.
- Проєктування бази даних: створити структуру для зберігання точок інтересу, маршрутів та історії користувача.
- Інтеграція картографічних API: забезпечити взаємодію з Google Maps API для роботи з картами, побудови маршрутів та геокодування.
- Реалізація системи: створити функціональний інтерфейс за допомогою .NET MAUI, що працюватиме на мобільних та настільних платформах.
- Тестування системи: перевірити стабільність, продуктивність і зручність використання додатку.

Основні функції системи:

- Відображення карт із можливістю інтерактивної взаємодії (масштабування, панорамування).
- Побудова оптимальних маршрутів між заданими точками.
- Пошук місць за адресою або категорією.

- Додавання та редагування точок інтересу (POI).
- Збереження та відображення історії маршрутів.
- Підтримка роботи з декількома користувачами.

Актуальність задачі:

У сучасному світі зростає попит на навігаційні додатки, які дозволяють ефективно вирішувати задачі маршрутизації, зберігання та обробки географічних даних. Картографічні системи стають основою для багатьох сфер — від туризму до логістики. Наявні рішення часто обмежені у функціональності, або мають складні інтерфейси, що ускладнює їхнє використання.

Розробка такого додатку дозволить:

Задовольнити потреби користувачів у швидкому доступі до картографічних даних.

Оптимізувати процеси побудови маршрутів.

Надати сучасний, інтуїтивно зрозумілий інтерфейс.

Сутність постановки задачі полягає у створенні навігаційного додатку, який максимально ефективно працює з картографічними API, зберігає необхідні дані у базі, а також забезпечує просту й швидку взаємодію з користувачем.

Рисунок 1.1 - концептуальна схему роботи системи

1.1.1.Функціональні вимоги

Система для роботи з навігаційними додатками повинна забезпечувати зручність, швидкість та точність обробки даних. Основні функціональні вимоги до навігаційного додатку, який використовує .NET MAUI та картографічні API, наведені нижче:

Коректне виконання функцій:

Кожна функція повинна бути реалізована відповідно до вимог і виконувати поставлені задачі без помилок.

Робота з картографічними API:

Інтеграція з Google Maps API або іншими подібними сервісами для відображення карт, побудови маршрутів та геокодування адрес.

1. Пошук об'єктів:

Можливість пошуку точок інтересу (POI) за назвою, адресою або категорією.

Швидкість пошуку не повинна перевищувати 1 секунду.

2. Побудова маршрутів:

Побудова оптимальних маршрутів між декількома точками з урахуванням часу та відстані.

Динамічне оновлення маршруту в разі зміни умов (наприклад, трафіку).

3. Обробка даних користувачів:

Підтримка декількох облікових записів користувачів.

Збереження історії маршрутів кожного користувача в базі даних.

4. Робота з базою даних:

Використання SQLite для зберігання даних про маршрути, точки інтересу та налаштування користувача.

Забезпечення швидкого доступу до бази даних із часом відповіді не більше 0,2 секунди.

5. Інтерактивність карти:

Можливість масштабування, панорамування карти та додавання маркерів.

6. Звіти та аналітика:

Формування звітів про використання маршрутів за заданий період.

Відображення статистики про популярні точки інтересу.

7. Додаткові можливості:

Можливість завантаження карт для роботи в офлайн-режимі.

Автоматичне оновлення даних про трафік в реальному часі.

Дані вимоги спрямовані на забезпечення надійної роботи системи, яка відповідає сучасним стандартам і потребам користувачів.

1.1.2.Нефункціональні вимоги

Нефункціональні вимоги описують, як система повинна виконувати свої функції, забезпечуючи якість роботи та зручність використання. До ключових вимог для навігаційного додатку належать:

- Інтерфейс користувача (UI/UX):
  - Інтерфейс повинен бути лаконічним і зрозумілим для користувачів, незалежно від їхнього рівня технічної підготовки.

  - Карта та інші елементи повинні коректно масштабуватися для різних екранів (мобільних, планшетів, настільних ПК).
  - Усі дії (пошук, побудова маршруту, взаємодія з картою) повинні бути доступними в декілька кліків.
- Швидкодія:
  - Час завантаження картографічного інтерфейсу не повинен перевищувати 2 секунд.
  - Побудова маршруту повинна займати не більше 1 секунди.
  - Реакція на дії користувача (наприклад, масштабування карти) повинна бути миттєвою.
- Надійність:
  - Додаток повинен стабільно працювати за умов обмеженого інтернет-з'єднання, з можливістю роботи в офлайн-режимі.
  - Дані (маршрути, точки інтересу) мають бути збережені без втрат у випадку неочікуваного завершення роботи додатку.
- Масштабованість:
  - Система повинна підтримувати можливість розширення функціоналу, наприклад, додавання нових картографічних сервісів або аналітичних модулів.
  - Архітектура має дозволяти одночасну роботу великої кількості користувачів без погіршення продуктивності.
- Кросплатформеність:
  - Додаток повинен коректно працювати на всіх підтримуваних платформах (Android, iOS, Windows, macOS) завдяки використанню .NET MAUI.
- Безпека:
  - Всі дані користувачів (наприклад, точки інтересу чи історія маршрутів) повинні бути зашифровані.
  - Впровадження аутентифікації користувачів для захисту персональної інформації.
- Доступність:

- Додаток повинен відповідати стандартам доступності, включаючи можливість використання людьми з обмеженими можливостями (наприклад, підтримка екранних рідерів).

- Ефективність використання ресурсів:
  - Додаток повинен бути оптимізований для економії батареї на мобільних пристроях та зменшення використання оперативної пам’яті.
- Підтримка локалізації:
  - Додаток має підтримувати декілька мов інтерфейсу (наприклад, українську та англійську), щоб забезпечити його зручність для різних користувачів.

Ці нефункціональні вимоги допоможуть забезпечити якісну та стабільну роботу навігаційного додатку, а також зручність для кінцевих користувачів.

1.2.Розгляд аналогів ПЗ

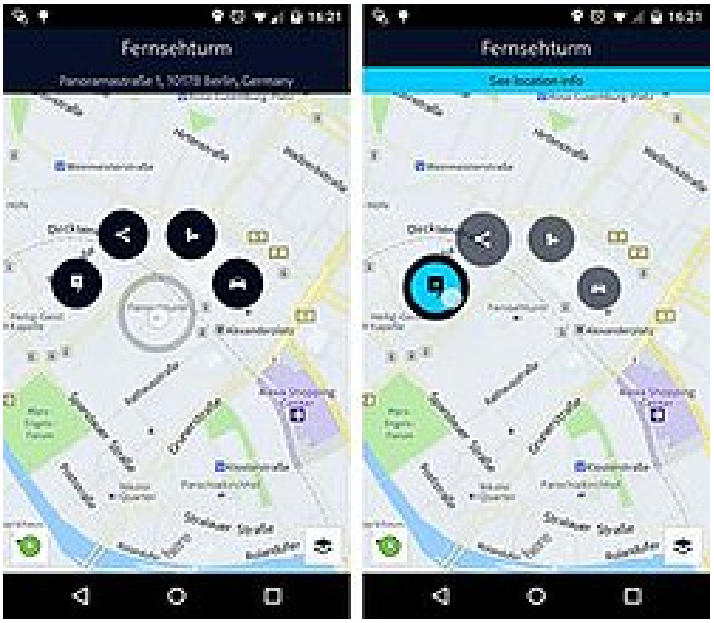

Рисунок 1.2 – Here WeGo

У рамках дослідження та розробки багатоплатформного навігаційного

додатку було проведено аналіз існуючих програмних рішень у цій сфері. Мета аналізу полягала у виявленні переваг, недоліків та основних функціональних можливостей популярних аналогів для визначення оптимального підходу до створення власного рішення. Розглянемо найвідоміші навігаційні додатки, що займають провідні позиції на ринку.

Here WeGo

Розробник: Here Technologies

Опис: Цей додаток відомий своїми можливостями офлайн-навігації, а також детальним відображенням громадського транспорту. Його часто використовують у регіонах із низьким рівнем доступу до інтернету [14].

Переваги:

- Розширені можливості офлайн-навігації.
- Інтеграція з інформацією про громадський транспорт.
- Висока деталізація карт у певних регіонах.

Недоліки:

- Обмежена функціональність у краудсорсингових даних.
- Інтерфейс менш інтуїтивний у порівнянні з конкурентами.
- Повільне оновлення дорожніх даних у реальному часі.

1.3.Вибір моделі розробки

Для розробки навігаційного додатку було обрано ітеративно-інкрементну модель розробки. Ця модель дозволяє розбити процес створення додатку на окремі ітерації, кожна з яких завершується додаванням функціонального компоненту до системи [5]. Основними причинами вибору цієї моделі є гнучкість

та можливість поступового вдосконалення додатку відповідно до вимог користувачів та змін у середовищі розробки

Причини вибору ітеративно-інкрементної моделі:

1. Гнучкість:

Модель дозволяє змінювати вимоги та функціонал системи на будь-якому етапі розробки без значних втрат часу і ресурсів.

2. Поетапна реалізація:

Проєкт поділяється на кілька фаз (ітерацій), кожна з яких зосереджується на розробці окремих функцій (наприклад, інтеграція картографічного API, створення бази даних, тестування інтерфейсу).

3. Рання демонстрація результатів:

Уже після перших ітерацій можна отримати робочу версію продукту, що дозволяє оцінити його функціональність і відповідність вимогам.

4. Зниження ризиків:

Виявлення проблем на ранніх етапах розробки дозволяє швидко усунути їх і уникнути значних витрат у майбутньому.

5. Підвищення якості:

Кожна ітерація завершується тестуванням, що дозволяє поступово вдосконалювати систему та гарантувати її стабільність.

Особливості реалізації моделі в рамках проєкту:

6. Планування:

На першій ітерації визначаються базові функції додатку (наприклад, робота з картами та побудова маршрутів).

У наступних ітераціях додаються нові компоненти, такі як інтерактивність карти, підтримка користувачів, збереження історії маршрутів тощо.

7. Розробка прототипу:

Початкова ітерація передбачає створення базового прототипу з мінімальним функціоналом для оцінки дизайну та архітектури.

8. Інтеграція компонентів:

У кожній наступній ітерації додаток доповнюється новими компонентами,

які інтегруються у вже існуючу систему.

9. Тестування:

Після кожної ітерації проводиться ретельне тестування для забезпечення коректної роботи системи.

Очікувані результати від застосування моделі:

- Зниження ризиків завдяки ранньому виявленню проблем.
- Гнучкість у зміні вимог на пізніх етапах розробки.
- Поступове покращення функціональності та стабільності додатку.

Ця модель розробки є ідеальним вибором для навігаційного додатку, оскільки вона забезпечує гнучкість, швидку адаптацію до змін і дозволяє отримати якісний продукт у найкоротші терміни.

1.4.Вибір технологій та інструментів для розробки

1.4.1.Мова програмування C#

C# - це сучасна мова програмування, розроблена компанією Microsoft у 2000 році для роботи в середовищі .NET. Вона є однією з найпопулярніших мов завдяки своїй універсальності, легкості у вивченні та потужності. C# базується на принципах об’єктно-орієнтованого програмування, що дозволяє ефективно структурувати код, розбивати його на модулі та повторно використовувати компоненти. Це забезпечує зручність у створенні різноманітних типів програмного забезпечення: веб-додатків, мобільних застосунків, настільних програм, серверних систем та ігор.

Однією з ключових переваг C# є її інтеграція з платформою .NET, яка надає широкий набір інструментів і бібліотек для розробки програмного забезпечення. Завдяки платформі .NET C# забезпечує автоматичне керування пам’яттю через механізм збирача сміття (Garbage Collector), що дозволяє уникати проблем із витоками пам’яті. Крім того, C# підтримує велику кількість вбудованих

типів даних і механізмів для роботи з ними, таких як масиви, списки, словники та інші колекції.

Мова C# має багатий синтаксис, який дозволяє писати ефективний та читабельний код. Вона підтримує базові концепції об’єктно-орієнтованого програмування, такі як інкапсуляція, спадкування та поліморфізм. Наприклад, у C# можна створювати класи, які містять властивості, методи та події, а також використовувати інтерфейси для забезпечення гнучкості архітектури. Завдяки цьому розробка великих проектів стає більш впорядкованою та зрозумілою.

C# підтримує роботу з різними типами додатків. Веб-додатки можна створювати за допомогою ASP.NET, яка забезпечує розробку динамічних сайтів та веб-служб. Для настільних програм використовуються такі інструменти, як Windows Forms, WPF (Windows Presentation Foundation) або MAUI (Multi-platform App UI), які дозволяють створювати додатки з графічним інтерфейсом для Windows та інших платформ. Завдяки Xamarin або MAUI можна розробляти мобільні додатки для iOS та Android, використовуючи спільну базу коду. В ігровій індустрії C# є основною мовою для Unity – популярного ігрового рушія, який дозволяє створювати ігри для різних платформ.

Одна з сильних сторін C# – підтримка LINQ (Language Integrated Query), яка дозволяє писати запити до колекцій даних, баз даних або XML у зручній формі. LINQ робить код більш компактним і зрозумілим, дозволяючи обробляти дані на високому рівні абстракції. Наприклад, замість багаторядкових циклів можна написати кілька зрозумілих рядків коду для фільтрації чи сортування даних.

C# також має широкий набір засобів для роботи з асинхронними операціями, що дозволяє створювати додатки, які ефективно працюють із запитами до зовнішніх сервісів чи виконують тривалі обчислення без блокування основного потоку. Асинхронність реалізована через ключові слова async та await, які значно спрощують написання коду порівняно з іншими мовами.

Мова C# активно використовується в корпоративному програмуванні

завдяки її можливостям інтеграції з іншими технологіями, такими як бази даних, веб-служби та хмарні платформи. Наприклад, C# часто використовується для створення серверних систем, які взаємодіють із базами даних через Entity Framework або інші ORM-бібліотеки. Крім того, C# дозволяє легко створювати RESTful API, які можуть обслуговувати мобільні та веб-додатки.

Завдяки багатству функцій, розвинутій екосистемі інструментів і бібліотек, а також підтримці від Microsoft [1], C# залишається однією з найпопулярніших мов програмування у світі. Вона підходить як для новачків, які тільки починають свій шлях у програмуванні, так і для професіоналів, які шукають інструменти для створення складних і масштабованих систем.

1.4.2.Фреймворк .Net

.NET — це універсальна платформа для розробки програмного забезпечення, яка дозволяє створювати додатки для різних пристроїв та операційних систем, включаючи Windows, macOS, Linux, Android та iOS. Вона базується на об’єднаній архітектурі, яка забезпечує високу продуктивність, кросплатформенність, модульність і гнучкість у використанні. Основою платформи є .NET Runtime [3], який виконує коди програм, управляє пам’яттю, потоками і обробляє винятки. Він складається з Common Language Runtime (CLR) та його спрощеної версії CoreCLR, яка використовується для кросплатформних проєктів.

Базовий набір функціональності забезпечується бібліотеками класів, відомими як Base Class Library (BCL). Ці бібліотеки містять все необхідне для роботи з основними типами даних, файловою системою, мережею, безпекою, багатопотоковістю та іншими фундаментальними функціями. Вони створені таким чином, щоб бути уніфікованими для всіх підтримуваних платформ, що дозволяє розробникам використовувати однаковий код незалежно від середовища

виконання. Для підтримки сучасних підходів до роботи з даними в платформу інтегровано LINQ, який дозволяє писати запити до колекцій і баз даних безпосередньо в коді.

.NET SDK включає всі необхідні інструменти для створення, тестування та запуску додатків. До нього входить компілятор Roslyn, який підтримує сучасні можливості мови C#, і командний інтерфейс CLI, що дозволяє створювати проєкти, збирати їх і запускати з командного рядка. Шаблони проєктів спрощують початок роботи з різними типами додатків, включаючи веб-додатки, мобільні додатки, десктопні програми та бібліотеки класів.

Для створення різних типів додатків у .NET доступні спеціалізовані фреймворки. Наприклад, ASP.NET Core використовується для розробки високопродуктивних веб-додатків та API, Entity Framework Core дозволяє зручно працювати з базами даних, а .NET MAUI забезпечує створення багатоплатформних мобільних і десктопних додатків із єдиним кодом для інтерфейсу користувача. Для роботи з десктопними додатками під Windows можна використовувати WPF або Windows Forms. Багатоплатформність платформи забезпечується можливістю виконання одного і того ж додатку на різних операційних системах із мінімальними змінами.

.NET також має тісну інтеграцію [4] з хмарними сервісами, такими як Microsoft Azure, що дозволяє розробляти та розгортати сучасні хмарні рішення. Завдяки підтримці контейнерів та інтеграції з Docker і Kubernetes розробники можуть легко створювати, тестувати та масштабувати додатки в хмарних середовищах. Для спрощення управління залежностями в платформу інтегровано NuGet, що забезпечує зручний доступ до тисяч бібліотек, необхідних для розробки.

Розробницькі середовища, такі як Visual Studio, Visual Studio Code або Rider, забезпечують зручність роботи з платформою .NET. Вони надають інструменти для аналізу коду, рефакторингу, дебагінгу та тестування, що значно підвищує ефективність розробки.

Узагальнюючи, структура .NET забезпечує універсальність, модульність і масштабованість. Вона підходить для створення як простих консольних додатків,

так і складних систем, включаючи веб-додатки, мобільні рішення, хмарні сервіси та багато іншого. Завдяки уніфікованій архітектурі та потужним інструментам платформа є одним із найкращих виборів для сучасних розробників програмного забезпечення.

1.4.3. NET MAUI як технологія для розробки клієнтських додатків

.NET MAUI (Multi-platform App UI) — це сучасна кросплатформна фреймворк від Microsoft для створення інтерфейсів користувача (UI), який дозволяє розробляти додатки для Android, iOS, Windows та macOS за допомогою єдиного базового коду. Ця технологія є еволюційним продовженням Xamarin.Forms і забезпечує потужний набір інструментів для розробки інтерактивних та зручних користувацьких інтерфейсів.

Основні переваги .NET MAUI включають його кросплатформність, що дозволяє створювати один код для додатків, які працюють на Android, iOS, Windows та macOS. Це значно зменшує витрати часу на підтримку і оновлення додатків для кількох платформ, спрощуючи процес розробки та підтримки. Гнучка архітектура .NET MAUI підтримує патерн MVVM, який забезпечує чітке розділення логіки бізнес-процесів і відображення інтерфейсу користувача, що полегшує підтримку і тестування додатку. Крім того, .NET MAUI дозволяє інтегрувати сучасні API, такі як Google Maps API, що є особливо важливим для роботи з картографічними даними.

Інтерфейс у .NET MAUI створюється за допомогою XAML, мови розмітки, яка дозволяє зручно і швидко розробляти UI. Бібліотека компонентів надає широкий набір елементів, таких як кнопки, текстові поля, списки, карти та інші елементи інтерфейсу. Завдяки ефективній взаємодії з платформами, специфічний функціонал кожної з них можна реалізувати через Dependency Service або

платформоспецифічні розширення. Це дозволяє адаптувати додаток до особливостей кожної операційної системи.

Продуктивність .NET MAUI базується на використанні потужності рідних платформ через оптимізований доступ до API. Це забезпечує швидке виконання додатків та мінімізацію затримок, що важливо для користувачів, які очікують плавної і швидкої роботи додатку.

Для навігаційних додатків .NET MAUI має додаткові можливості. Серед них інтеграція з картографічними API, такими як Google Maps, Mapbox та OpenStreetMap, що забезпечує роботу з картами, побудову маршрутів та геокодування. Легка робота з платформоспецифічними елементами дозволяє створювати індивідуальний інтерфейс для різних платформ за допомогою розширень. Система також підтримує офлайн-функціонал завдяки можливості зберігати дані у локальній базі SQLite, що дозволяє додатку працювати без доступу до інтернету. Адаптивний дизайн .NET MAUI автоматично підлаштовується під різні розміри екранів, включаючи мобільні пристрої, планшети та настільні ПК, що робить його універсальним вибором для розробників, які прагнуть створювати сучасні, продуктивні та зручні для користувачів додатки.

Чому .NET MAUI підходить для цієї роботи

- Кросплатформність дозволяє створити додаток, що працює на різних пристроях, з мінімальними витратами.
- Інтеграція API спрощує роботу з картографічними даними.
- Зручність і сучасність UI/UX забезпечують легкість використання додатку навіть для користувачів із мінімальним досвідом.
- Можливість роботи офлайн робить систему більш гнучкою та незалежною від постійного підключення до мережі.

.NET MAUI — це сучасний інструмент для створення масштабованих, інтерактивних та зручних у використанні навігаційних додатків.

1.4.4.MVVM (Model-View-ViewModel) Архітектурний патерн

MVVM (Model-View-ViewModel) — це архітектурний патерн, призначений для розробки програм з графічним інтерфейсом (GUI), який забезпечує чітке розділення бізнес-логіки, інтерфейсу користувача (UI) та механізмів взаємодії між ними. MVVM є основним патерном у .NET MAUI та широко використовується для створення кросплатформенних додатків.

Основними компонентами архітектурного патерну MVVM є Model, View і ViewModel. Model відповідає за обробку даних та реалізацію бізнес-логіки. Вона включає логіку доступу до бази даних, інтеграцію з API та правила валідації даних. Модель є незалежною від інтерфейсу користувача і не має уявлення про нього. View відповідає за відображення даних та взаємодію з користувачем. У .NET MAUI це зазвичай XAML-файли, які містять декларативний опис інтерфейсу. View не містить бізнес-логіки і працює лише з ViewModel через механізм прив’язок. ViewModel є "мостом" між Model і View. Вона забезпечує підготовку даних для View та реагує на дії користувача. ViewModel не має прямого доступу до View, і комунікація між ними здійснюється через прив’язки.

Серед основних механізмів MVVM можна виділити прив’язку даних, команди та використання інтерфейсу INotifyPropertyChanged. Прив’язка даних забезпечує двосторонній зв’язок між View та ViewModel, що дозволяє автоматично синхронізувати дані між інтерфейсом і логікою. Наприклад, якщо користувач вводить текст у полі, це автоматично оновлює відповідну властивість у ViewModel і навпаки. Команди використовуються для обробки подій, таких як кліки на кнопки, і замінюють традиційний підхід до обробки подій у UI. Команди реалізуються через інтерфейс ICommand. Інтерфейс INotifyPropertyChanged дозволяє ViewModel повідомляти View про зміни у властивостях, що є основним механізмом для роботи з прив’язками.

Серед переваг MVVM можна виділити чітке розділення відповідальностей,

тестованість, повторне використання коду та гнучкість. Чітке розділення відповідальностей означає, що ViewModel відповідає за логіку, а View — лише за відображення. Це спрощує підтримку та масштабування додатку. Логіка, реалізована у ViewModel, легко тестується, оскільки вона не залежить від View. Також ViewModel може використовуватись у кількох різних View, а логіка, яка реалізована у Model, є універсальною та незалежною від інтерфейсу. Зміни у View або ViewModel не впливають на Model, що полегшує внесення змін до системи. У підсумку, MVVM забезпечує зручне розділення обов'язків, дозволяє легко змінювати інтерфейс, повторно використовувати код і підтримувати модульність додатку. Цей патерн ідеально підходить для роботи з .NET MAUI, зокрема для навігаційних додатків, де важлива інтеграція з API, динамічний інтерфейс та тестованість логіки. Реалізація ентерпрайз-патерну MVVM продемонстрована в архітектурі системи.

1.4.5.Середовище розробки JetBrains Rider

Інтегроване середовище розробки JetBrains Rider - це креативний інструмент, який можна використовувати для редагування, налагодження, створення програмних продуктів та їх публікації. Інтегроване середовище розробки (IDE) - це багатофункціональна програма, яка може бути використана для різноманітних напрямків розробки програмного забезпечення. Окрім стандартного редактора та налагоджувача, який надає більшість IDE, тут ще включені компілятори, інструменти для форматування та доповнення коду, графічні середовища та інші функції які значно полегшують процес розробки програмного забезпечення [5].

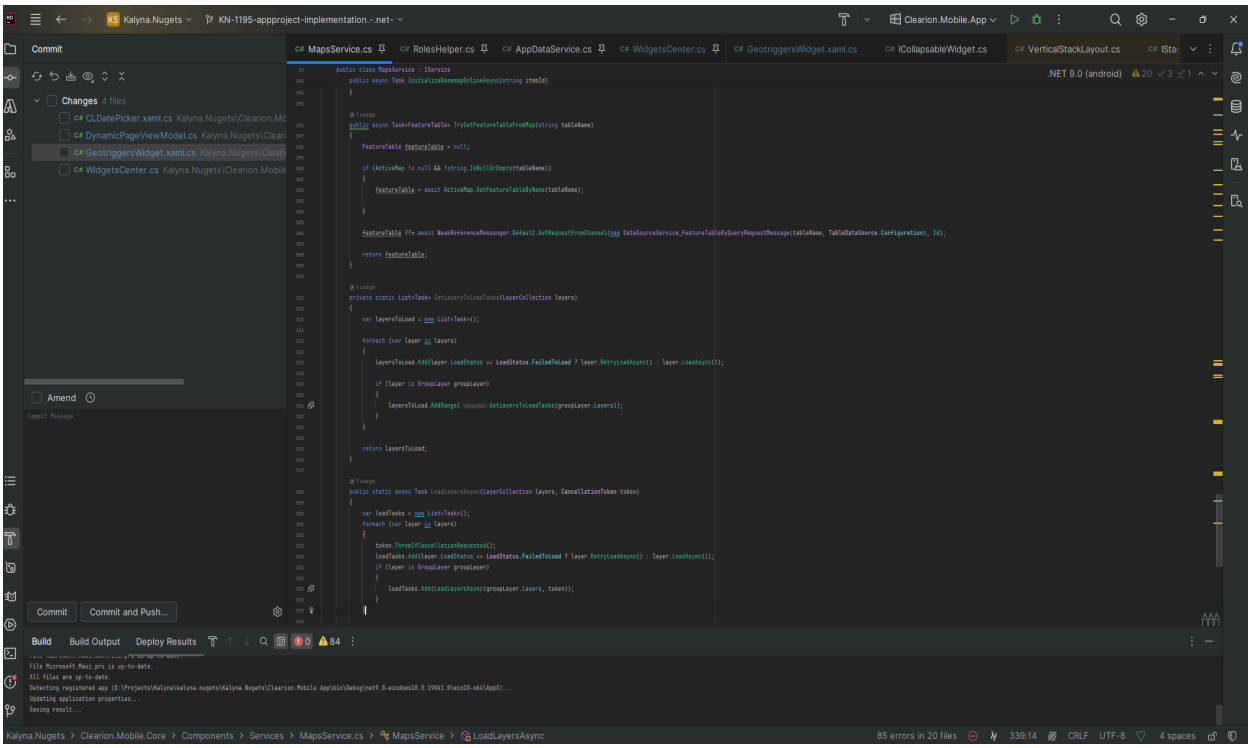

Рисунок 1.3 – Середовище розробки JetBrains Rider

На рисунку 1.3 зображено JetBrains Rider з відкритим проектом та кількома ключовими вікнами інструментів, Solution Explorer (угорі ліворуч) дозволяє переглядати, переміщуватися та керувати файлами коду. Провідник рішень може допомогти впорядкувати код, згрупувавши файли у рішення та проекти. Дане середовище розробки підтримує широкий спектр мов програмування, таких як C#, VB.NET, F#, JavaScript, TypeScript, HTML, CSS, та інші. JetBrains Rider забезпечує інтеграцію з платформою .NET, а також підтримує роботу з популярними фреймворками, такими як ASP.NET, Xamarin, Unity, що робить його універсальним інструментом для різних типів проектів.

У правій частині зображення видно вікно Code Editor, яке надає можливість зручно писати та редагувати код. Воно підтримує підсвічування синтаксису, автозаповнення та аналіз коду в реальному часі, що значно полегшує роботу розробника. Завдяки вбудованому механізму підказок та рефакторингу, середовище допомагає оптимізувати код та забезпечити його читабельність.

У нижній частині зображено Terminal та Debugging Tools, які дозволяють

виконувати налагодження, перевірку та запуск проектів безпосередньо в IDE. Rider підтримує потужні засоби для відстеження помилок і діагностики, включаючи візуалізацію стеку викликів, відображення змінних і контроль точок зупинки.

Середовище JetBrains Rider також інтегрується з системами контролю версій, такими як Git, що дає можливість розробникам ефективно керувати змінами у коді та співпрацювати з іншими учасниками команди. Його функціонал робить роботу з великими проектами більш організованою і продуктивною.

1.4.6.База даних MySql

MySQL є однією з найпоширеніших систем керування реляційними базами даних у світі. Вона була створена для забезпечення швидкого, надійного і простого у використанні середовища для роботи з базами даних. MySQL використовується як для невеликих веб-додатків, так і для великих корпоративних проектів, завдяки своїй масштабованості та можливості працювати з великими обсягами даних. Відкрита ліцензія дозволяє користувачам вільно використовувати її у своїх проектах, що зробило MySQL однією з найпопулярніших технологій для роботи з базами даних.

MySQL базується на реляційній моделі, де дані організовані у вигляді таблиць. Це дозволяє розробникам створювати структуровані зв'язки між таблицями за допомогою ключів, таких як первинні і зовнішні ключі. Ця особливість спрощує організацію даних і забезпечує їх логічну цілісність. Однією з основних причин популярності MySQL є її підтримка мови SQL (Structured Query Language), яка дозволяє взаємодіяти з базами даних через стандартні запити, такі як SELECT, INSERT, UPDATE та DELETE.

Серед важливих особливостей MySQL можна виділити підтримку транзакцій. Використання стандарту ACID (атомарність, цілісність, ізоляція, довговічність) забезпечує надійність і цілісність даних навіть у разі збою.

Наприклад, якщо операція з даними була виконана частково через технічний збій, система може повернутися до початкового стану, гарантуючи, що жодні дані не будуть втрачені чи пошкоджені. Це особливо важливо для банківських систем, систем електронної комерції та інших додатків, де точність і надійність є критичними.

MySQL також відома своєю швидкодією. Вона оптимізована для обробки великої кількості запитів у реальному часі, що робить її ідеальним вибором для веб-додатків, які повинні обслуговувати тисячі користувачів одночасно. Для забезпечення ще більшої продуктивності MySQL підтримує реплікацію баз даних, яка дозволяє створювати копії даних на різних серверах. Це забезпечує високу доступність і дозволяє балансувати навантаження між кількома серверами.

MySQL також відзначається своєю гнучкістю. Вона підтримує багато операційних систем, таких як Windows, Linux, macOS і UNIX, що дозволяє розробникам вибирати платформу відповідно до своїх потреб. Крім того, MySQL підтримує багато мов програмування, включаючи PHP, Python, Java, C# і C++, що робить її універсальною для різних проектів.

Ще однією важливою особливістю MySQL є її механізми зберігання даних. Найбільш поширеними є InnoDB і MyISAM. InnoDB підтримує транзакції, індекси та забезпечує високу надійність. Він підходить для додатків, де важливі цілісність даних і багатокористувацький доступ. MyISAM забезпечує швидке виконання запитів і підходить для додатків, де транзакції не є критичними, наприклад, для аналізу великих обсягів даних.

Безпека також є важливим аспектом MySQL. Вона підтримує аутентифікацію користувачів, шифрування даних і контроль доступу. Наприклад, адміністратор може обмежити доступ до певних таблиць або баз даних для конкретних користувачів, що дозволяє захищати конфіденційні дані. Також доступні функції для створення резервних копій і відновлення даних, що забезпечує додатковий рівень захисту.

Одним із важливих інструментів для роботи з MySQL є MySQL Workbench. Це графічний інтерфейс, який дозволяє розробникам проектувати структуру баз даних, виконувати SQL-запити, аналізувати продуктивність і налагоджувати

роботу бази. MySQL Workbench спрощує роботу навіть для новачків, оскільки надає зручний візуальний інструментарій.

MySQL має низку переваг, серед яких відкритість коду, висока продуктивність, велика спільнота розробників, а також регулярні оновлення і підтримка від Oracle. Однак, як і будь-яка технологія, MySQL має свої обмеження. Наприклад, вона поступається деяким іншим СКБД, таким як PostgreSQL, у роботі зі складними типами даних і розширеними SQL-функціями. Однак її простота, ефективність і універсальність роблять MySQL ідеальним вибором для більшості проектів, особливо тих, які потребують високої продуктивності та надійності.

1.4.7.SQLite в .NET MAUI

SQLite в .NET MAUI є ключовим інструментом для організації локального зберігання даних. Це легковагова реляційна база даних, яка працює без необхідності встановлення окремого серверу, що робить її ідеальним рішенням для мобільних і багатоплатформних додатків. У MAUI SQLite інтегрується через бібліотеки, такі як SQLite-net, що дозволяє розробникам працювати з базою даних за допомогою об’єктно-реляційного відображення (ORM). Це значно спрощує взаємодію з даними, оскільки більшість операцій, таких як створення таблиць, виконання запитів або оновлення записів, можна реалізувати за допомогою об’єктів і методів, замість написання сирих SQL-запитів.

SQLite зберігає дані у вигляді одного файлу, розташованого в спеціальній директорії додатка. Для мобільних платформ, таких як Android і iOS, цей файл зазвичай розміщується у внутрішньому сховищі додатка, що забезпечує безпеку і захищеність даних від стороннього доступу. Для Windows і macOS цей файл можна зберігати в папці, доступній для користувача, або в будь-якому іншому визначеному каталозі. Завдяки своїй компактності та мінімальним вимогам до ресурсів, SQLite не створює додаткового навантаження на систему, що є важливим

для мобільних пристроїв із обмеженою пам’яттю.

SQLite забезпечує підтримку всіх основних функцій реляційних баз даних, включаючи операції CRUD (створення, читання, оновлення, видалення), транзакції та використання індексів для оптимізації пошуку. У поєднанні з .NET MAUI, розробники можуть створювати багатоплатформні додатки з єдиною структурою бази даних, що значно знижує складність коду та підвищує продуктивність розробки. Наприклад, розробник може створити модель даних у вигляді C# класу, а бібліотека SQLite-net автоматично зв’яже її з таблицею в базі даних. Це дозволяє уникнути ручного налаштування таблиць або написання складних SQL-запитів для кожної операції.

Однією з ключових переваг використання SQLite у .NET MAUI є її сумісність із різними платформами. Це означає, що, розробивши один раз структуру бази даних і код для роботи з нею, розробник може використовувати його на всіх підтримуваних платформах, включаючи Android, iOS, Windows та macOS, без змін. Це досягається завдяки використанню спільного шару .NET MAUI, який надає універсальний доступ до базових функцій платформи.

SQLite часто використовується для зберігання конфігурацій додатка, даних користувача, кешування результатів веб-запитів, а також для реалізації складних функціональних можливостей, таких як офлайн-доступ до даних. Наприклад, додаток може синхронізувати дані з сервером, але при цьому зберігати їх локально для роботи у відсутності підключення до інтернету. Це робить SQLite ідеальним вибором для додатків, які працюють у нестабільних мережевих умовах або повинні забезпечувати швидкий доступ до великих обсягів даних без необхідності постійного запиту до сервера.

Процес налаштування SQLite у MAUI починається зі створення моделі даних, яка зазвичай представляється у вигляді C# класів із властивостями, які відповідають колонкам у таблиці бази даних. Потім використовується бібліотека SQLite-net для створення або підключення до існуючої бази даних, виконання SQL-запитів і управління транзакціями. Розробник може використовувати LINQ-запити для виконання операцій із даними, що спрощує читання та підтримку коду. Це також дозволяє уникнути ризиків, пов’язаних із ручним

написанням SQL-запитів, таких як помилки синтаксису або SQL-ін’єкції.

Однією з важливих особливостей SQLite є її здатність працювати в офлайн-режимі. Це означає, що всі дані залишаються доступними навіть без підключення до мережі, а будь-які зміни можуть бути синхронізовані з сервером після відновлення підключення. Така функціональність особливо корисна для додатків, що використовуються у віддалених або сільських районах, де інтернет-з’єднання може бути обмеженим або нестабільним.

У .NET MAUI SQLite легко інтегрується з іншими сервісами та функціями платформи. Наприклад, вона може використовуватися разом із API для роботи з файлами, щоб створювати резервні копії бази даних, або з бібліотеками хмарного зберігання для завантаження й синхронізації даних між пристроями. Це відкриває широкі можливості для розробки функціонально багатих додатків із комплексними вимогами до зберігання й управління даними.

Таким чином, використання SQLite у .NET MAUI забезпечує розробникам потужний і гнучкий інструмент для роботи з локальними даними. Завдяки простоті використання, високій продуктивності й широкій сумісності, ця база даних є стандартом для багатьох мобільних і багатоплатформних додатків, що працюють із структурованими даними.

-

2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Проблематика та доцільність розробки

Картографічні сервіси відіграють важливу роль у сучасних навігаційних додатках, забезпечуючи користувачів інформацією про маршрути, трафік, геолокацію та інші аспекти. Однак інтеграція картографічних API у кросплатформні додатки, зокрема створені на основі .NET MAUI, має свої виклики. Основна проблематика полягає у складності одночасного забезпечення високої продуктивності, точної роботи функцій навігації та сумісності з різними платформами (iOS, Android, Windows).

По-перше, більшість популярних картографічних API, як-от Google Maps API, Mapbox чи Here Maps, пропонують потужні функціональні можливості, але їхня інтеграція в середовище .NET MAUI може бути ускладнена через обмежену кількість нативних бібліотек для роботи з картами, адаптованих для MAUI. Це вимагає використання додаткових обгорток (wrapper'ів) або створення власних кастомних рішень для взаємодії з платформозалежними компонентами. Наприклад, для роботи з Google Maps розробник повинен створювати окремі реалізації для Android та iOS, що додає складності в розробці та тестуванні.

По-друге, питання продуктивності є одним із ключових аспектів у цьому контексті. Картографічні сервіси часто обробляють великі обсяги даних у реальному часі, наприклад, при відображенні трафіку або побудові динамічних маршрутів. У кросплатформному середовищі продуктивність таких операцій може знижуватися через додаткові накладні витрати, викликані кросплатформною абстракцією MAUI. Важливим є також питання оптимізації роботи із запитами до API, які повинні бути максимально швидкими, щоб забезпечити плавний користувацький досвід.

Третім аспектом проблематики є складність інтеграції розширених функцій навігації. Багато картографічних API підтримують функції, як-от голосова навігація, відстеження поточного місця розташування, інтерактивна взаємодія з картами (додавання точок, маршрутів, зон тощо). Однак реалізація цих функцій у

MAUI вимагає значної роботи зі створення нативних рішень або кастомізації існуючих бібліотек. Особливо складно забезпечити однакову поведінку додатку на різних платформах, оскільки кожна з них має свої специфічні обмеження та можливості.

Ще одна важлива проблема стосується ліцензування та вартості використання картографічних API. Наприклад, Google Maps API має обмеження щодо безкоштовного використання, і перевищення лімітів призводить до додаткових витрат. Це ставить питання вибору найбільш оптимального сервісу з точки зору вартості та функціональних можливостей, особливо для проєктів із обмеженим бюджетом або навчальних.

Також варто зазначити, що інтеграція API потребує детального налаштування конфігурації безпеки, зокрема управління ключами доступу та обмеження прав доступу. Неправильна конфігурація може призвести до витоку даних або несанкціонованого використання API сторонніми особами, що є критично важливим для додатків із конфіденційними даними.

Таким чином, проблематика дослідження включає цілу низку технічних, продуктивних, фінансових і навчальних викликів, які виникають при інтеграції картографічних API у .NET MAUI-додатки. Їхнє вирішення є важливим для забезпечення успішної реалізації навігаційного функціоналу, що є ключовою метою сучасних мобільних додатків.

Необхідність твоєї розробки обумовлена кількома ключовими факторами. По-перше, сучасний світ вимагає ефективних навігаційних рішень, які можуть працювати на різних платформах і забезпечувати широкий функціонал. Попит на мобільні додатки, які допомагають користувачам знаходити оптимальні маршрути, уникати заторів, планувати подорожі з урахуванням індивідуальних вподобань, стрімко зростає. При цьому важливою є кросплатформність таких рішень, яка дозволяє зменшити витрати на розробку і забезпечити підтримку різних операційних систем.

По-друге, .NET MAUI пропонує нові можливості для розробників у створенні кросплатформних додатків, однак його інтеграція з картографічними

API залишається недостатньо вивченою. Це створює виклик для розробників, які прагнуть реалізувати функціонал карт та навігації з використанням сучасних підходів. Твоя робота спрямована на закриття цього пробілу і може стати корисним прикладом для інших фахівців.

Крім того, інтеграція картографічних API є складним процесом, який вимагає не тільки знання особливостей кожного сервісу, але й ефективної реалізації функцій маршрутизації, обробки трафіку та інтерактивної взаємодії з картою. Це особливо важливо для додатків, орієнтованих на туристичну сферу, де якісна навігація є одним із головних критеріїв користувацького задоволення.

Ще одним важливим аспектом є необхідність забезпечення персоналізації та управління користувачами, що включає реєстрацію, автентифікацію та збереження даних. Використання AWS Cognito у твоїй розробці дозволяє вирішити ці завдання без суттєвих витрат на створення власної інфраструктури.

Зрештою, твоя розробка відповідає сучасним технологічним вимогам, задовольняє зростаючі потреби ринку навігаційних рішень і сприяє підвищенню доступності .NET MAUI як платформи для реалізації таких проектів. Це робить її актуальною не лише з практичної, але й з науково-дослідницької точки зору.

2.2 Управління користувачами

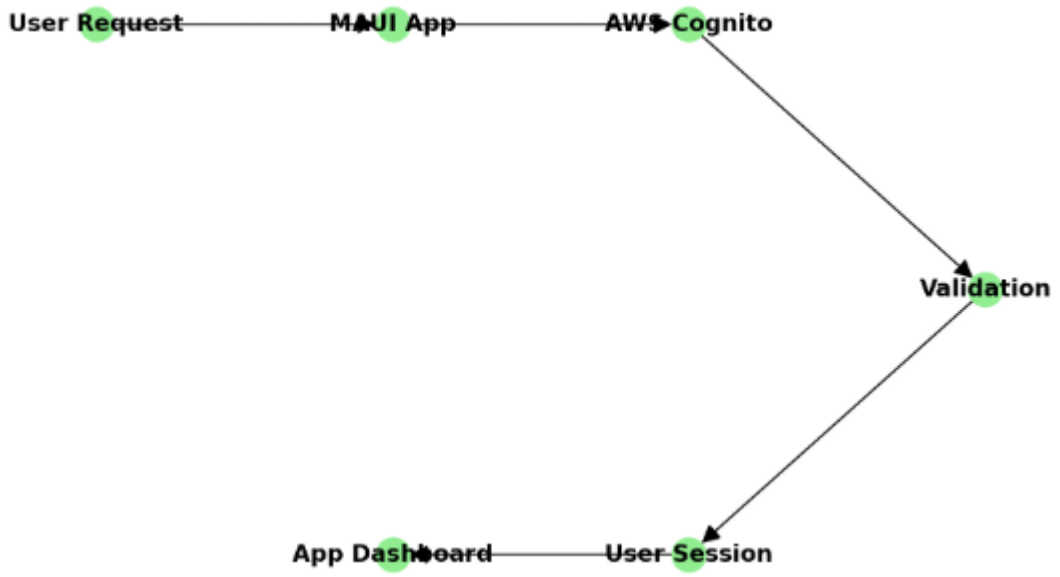

Рисунок 2.1 – Порядок роботи авторизації

Управління користувачами є одним із ключових аспектів навігаційного додатка, оскільки воно забезпечує персоналізований досвід, збереження індивідуальних налаштувань та безпеку даних. У цьому розділі розглядається процес імплементації функціоналу для роботи з користувачами, включаючи реєстрацію, авторизацію та автентифікацію, а також інтеграцію із зовнішніми сервісами для підтримки Single Sign-On (SSO). Особлива увага приділяється використанню AWS Cognito як основного інструменту для управління користувацькими обліковими записами у кросплатформному середовищі .NET MAUI.

Система розроблена на основі Node.js – популярної платформи для створення серверних додатків, що використовує неблокуючу модель вводу-виводу, що забезпечує високу продуктивність навіть при обробці великої кількості одночасних запитів.

Для реалізації інфраструктурних компонентів використано Amazon Web Services (AWS), що забезпечує масштабованість, надійність та безпеку зберігання даних. Завдяки інтеграції цих технологій, сервіс здатний ефективно виконувати завдання, пов'язані з аутентифікацією, авторизацією, реєстрацією користувачів і

управлінням їх даними.

Сервіс включає такі основні компоненти:

- Реєстрація користувачів:

Користувачі можуть створювати облікові записи, вводячи свої особисті дані. Верифікація може здійснюватися через email, SMS або використання SSO, що спрощує процес для кінцевого користувача.

- Авторизація та аутентифікація:

Використовується механізм токенів, наприклад JWT (JSON Web Tokens), для забезпечення безпечної сесії користувача. Токени передаються між клієнтом та сервером, дозволяючи перевірити права доступу до різних ресурсів.

- Single Sign-On (SSO):

Впровадження SSO дозволяє користувачам здійснювати одноразову аутентифікацію для доступу до кількох систем або додатків. Цей підхід реалізовано на основі протоколів OAuth 2.0 або OpenID Connect (OIDC). Підтримка соціальних мереж, таких як Google, Facebook, або корпоративних провайдерів (наприклад, Microsoft Active Directory), підвищує зручність для користувачів.

2.3 Розгортання бекенд-сервісу

Для локального розгортання бекенд сервісу і контейнеризації використовувався Docker. Цей підхід забезпечує стабільність, незалежність від середовища операційної системи та спрощує налаштування залежностей.

Перш за все, для розгортання сервісу необхідно створити образ Docker. Образ включає всі необхідні компоненти для роботи бекенду: платформу Node.js, залежності проєкту, а також конфігурацію для запуску серверу. Образ збирається на основі базового образу Node.js, який забезпечує стабільне середовище для виконання JavaScript-коду. До цього середовища додаються файли вашого проєкту

та встановлюються залежності, описані у файлі package.json. Завдяки цьому, під час запуску контейнера, додаток буде повністю готовий до роботи.

Наступним кроком є налаштування мережевих портів. У вашому сервісі ймовірно використовується порт 3000 для прийому HTTP-запитів, тому його потрібно відкрити в контейнері та прив'язати до локального порту комп'ютера. Це дозволить вам взаємодіяти з сервісом через браузер або інструменти для тестування API.

Для тестування роботи сервісу необхідно створити симуляцію середовища AWS. Це може бути реалізовано за допомогою таких інструментів, як Docker для розгортання та Postman для зручного тестування API [1].

Після запуску, бекенд-сервіс буде доступний за локальною адресою, наприклад http://localhost:3000. Для тестування API можна використовувати інструменти, такі як Postman, Insomnia або curl, щоб перевірити, чи правильно працюють маршрути, аутентифікація, реєстрація та інші функції.

Після завершення тестування важливо зупинити та видалити всі контейнери, щоб звільнити ресурси комп'ютера. Це легко виконується через команди Docker Compose.

Розгортання сервісу локально за допомогою Docker дозволяє імітувати реальне середовище роботи додатка, включаючи інтеграцію з хмарними сервісами AWS, що значно спрощує процес розробки, тестування та відлагодження.

2.5 Розробка мобільного додатку

Проєктування мобільного застосунку на .NET MAUI (Multi-platform App UI) включає кілька ключових етапів, кожен з яких забезпечує успішну реалізацію кросплатформного додатку, що працює на iOS, Android, Windows та macOS. MAUI дозволяє розробляти застосунок з єдиною базою коду, використовуючи мову C# та XAML для опису інтерфейсу користувача.

У процесі розробки застосунку я реалізував кілька ентерпрайз патернів проектування, які дозволили забезпечити масштабованість, зрозумілість та підтримуваність коду. Основою архітектури став патерн MVVM (Model-View-ViewModel), який успішно використаний для чіткого розділення логіки користувацького інтерфейсу та бізнес-логіки. Це забезпечило легкість у тестуванні та спростило внесення змін у функціональність без необхідності значних змін у структурі додатку.

Для зменшення жорстких зв’язків між компонентами було впроваджено патерни IoC (Inversion of Control) та Dependency Injection. Завдяки цим підходам стало можливим безперешкодно інтегрувати зовнішні сервіси, як-от AWS Cognito для автентифікації користувачів і Mapbox для роботи з картами. Застосування Dependency Injection також спростило заміну або оновлення компонентів, не впливаючи на основний код застосунку.

У реалізації складних бізнес-процесів я використав CQRS (Command Query Responsibility Segregation) для чіткого розділення запитів на отримання даних і команд для зміни стану. Це дало змогу зберігати чистоту коду та забезпечити легке масштабування при зростанні кількості користувачів. Додатково було впроваджено патерн Mediator, який дозволив централізувати обробку запитів і команд, мінімізуючи залежності між компонентами [2].

Такий підхід до архітектури та проектування додатку вже продемонстрував свою ефективність у вирішенні задач, пов'язаних із навантаженнями, змінами бізнес-вимог та інтеграцією нових функцій. Завдяки використанню цих патернів вдалося створити гнучке та надійне рішення, яке легко адаптується до майбутніх потреб.

2.5 Імплементація сервісу для навігації

Інтерфейс INavigationService призначений для організації навігації в додатку, що використовує архітектуру MVVM. Він абстрагує логіку переходів між ViewModel, що дозволяє зосередитися на бізнес-логіці без прив'язки до деталей реалізації навігації.

```
18 usages   1 inheritor   Vasyl Hlyva
public interface INavigationService
{
    7 usages   1 implementation   Vasyl Hlyva
    Task NavigateAsync<T>(object? parameter = null) where T : BaseViewModel;

    1 implementation   Vasyl Hlyva
    Task CloseAsync();
}
```

Рисунок 2.2 – Navigation Service API

Метод NavigateAsync<T>():

Призначення:

Використовується для переходу до View, асоційованого із ViewModel типу T.

Аргументи:

- T: Тип ViewModel, до якої потрібно здійснити перехід. Цей тип повинен успадковуватися від BaseViewModel, кожна з яких повинна бути зареєстрована в IoC контейнері.
- parameter: Опціональний параметр для передачі даних у ViewModel. Наприклад, це можуть бути дані, необхідні для відображення або обробки у новій ViewModel.

Результат:

- Повертає об'єкт Task, що дозволяє виконувати асинхронні операції під час переходу.

Імплементація інтерфейсу INavigationService інкапсулює в собі логіку знаходження сторінки та виклику методу PushAsync(). Дивитись додаток 1.

2.6. Обробка данних користувачів

Клас UserModel належить до простору імен SideQuestAdvisor.Models і є основною моделлю даних, яка представляє користувача в системі. Його структура включає різноманітні властивості, що забезпечують збереження інформації про користувача, його вподобання та історію взаємодії з атракціями.

Властивість PrefferedAttractions має тип List<AttractionModel> і використовується для збереження списку атракцій, які користувач віддає перевагу. Це може бути корисним для формування рекомендацій або відображення улюблених місць. Властивість RecentlyVisitedAttractions, також типу List<AttractionModel>, зберігає інформацію про атракції, які користувач нещодавно відвідав, що дозволяє вести історію взаємодій і, за потреби, пропонувати користувачу схожі місця.

Крім того, клас включає властивості FirstName і LastName, обидві типу string, які зберігають відповідно ім'я та прізвище користувача. Властивість Password, типу string, містить пароль користувача і позначена атрибутом [JsonProperty("password")], що вказує, як це поле має серіалізуватися і десеріалізуватися в JSON-форматі. Властивість Email, також типу string, зберігає адресу електронної пошти користувача і позначена атрибутом [JsonProperty("email")], який використовується аналогічним чином.

Ця модель є ключовою для збереження і передачі інформації про користувача, забезпечуючи взаємодію з іншими частинами системи, такими як бази даних, сервіси або інтерфейси API.

2.7 Опис та побудова маршруту

Клас RouteModel належить до простору імен SideQuestAdvisor.Models і слугує моделлю даних для представлення маршруту в додатку. Цей клас містить ключові властивості, які забезпечують опис маршруту, зокрема точку відправлення, пункт призначення та доступний час для виконання подорожі.

Властивість DeparturePoint має тип string і використовується для збереження початкового місця маршруту. Вона визначає, звідки починається шлях користувача і є вихідною точкою для побудови маршруту. Властивість DestinationPoint, також типу string, зберігає кінцевий пункт призначення маршруту. Це ключова інформація, яка дозволяє системі побудувати шлях до місця, куди користувач хоче дістатися.

Властивість AvailableTime має тип TimeSpan і визначає доступний час для завершення маршруту. Вона встановлює часові рамки, протягом яких користувач планує здійснити подорож. Це дозволяє системі адаптувати маршрут відповідно до обмежень часу, забезпечуючи максимально ефективне використання доступного періоду.

Клас RouteModel є важливим компонентом додатку, що відповідає за структурування даних маршруту, їх передачу між різними частинами системи, а також взаємодію з іншими модулями, наприклад, з сервісами побудови маршрутів або інтерфейсом користувача.

Призначення класу:

Клас RouteModel є DTO (Data Transfer Object), що використовується для передачі інформації про маршрут у різних частинах системи або між клієнтом та сервером.

Основне застосування:

1. Збереження даних маршруту для побудови оптимального шляху користувача.

2. Використання в навігаційних сервісах для розрахунку часу подорожі.
3. Формування запитів до картографічного API для побудови маршруту.

2.8 Алгоритм побудови маршрутів для користувача

Процес побудови маршрутів від точки A до точки B складається з кількох етапів, які враховують вподобання користувача та обмеження часу для надання найбільш релевантних варіантів маршрутів.

На першому етапі збираються всі необхідні дані, включаючи точку відправлення A, точку призначення B, вподобання користувача, такі як список атракцій, які він хоче відвідати (наприклад, парки чи музеї), а також обмеження часу, протягом якого маршрут має бути завершений. Ці дані формують основу для подальшої побудови маршруту.

Далі виконується запит до картографічного API. Сервіс отримує початкову інформацію та повертає кілька базових варіантів маршрутів між точками A і B. Зазвичай це найшвидший маршрут і кілька альтернативних варіантів, якщо вони доступні. Цей етап забезпечує вихідні дані для побудови користувацьких маршрутів.

На основі отриманих даних генеруються три різні маршрути. Перший маршрут є найшвидшим і обчислюється для мінімізації часу подорожі без додавання додаткових точок. Другий маршрут враховує вподобання користувача, включаючи точки інтересу, які відповідають його перевагам і знаходяться поблизу базового маршруту. Маршрут будується таким чином, щоб включити ці точки, але залишатися в межах доступного часу. Третій маршрут є найменш завантаженим. Для його побудови враховуються дані про завантаженість доріг у реальному часі, і вибір надається менш завантаженим шляхам, навіть якщо маршрут буде довшим за відстанню.

Після генерації трьох маршрутів виконується фільтрація. Всі маршрути перевіряються на відповідність обмеженням за часом. Якщо якийсь із них перевищує заданий час, він може бути відкинутий або оптимізований для скорочення тривалості.

На завершальному етапі користувачу надається три маршрути з описом їх основних характеристик, таких як час у дорозі, загальна відстань і список відвідуваних точок інтересу. Інформація подається у зручному форматі, що дозволяє користувачу легко обрати найбільш відповідний варіант для подорожі.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ:

3.1 Структура проекту

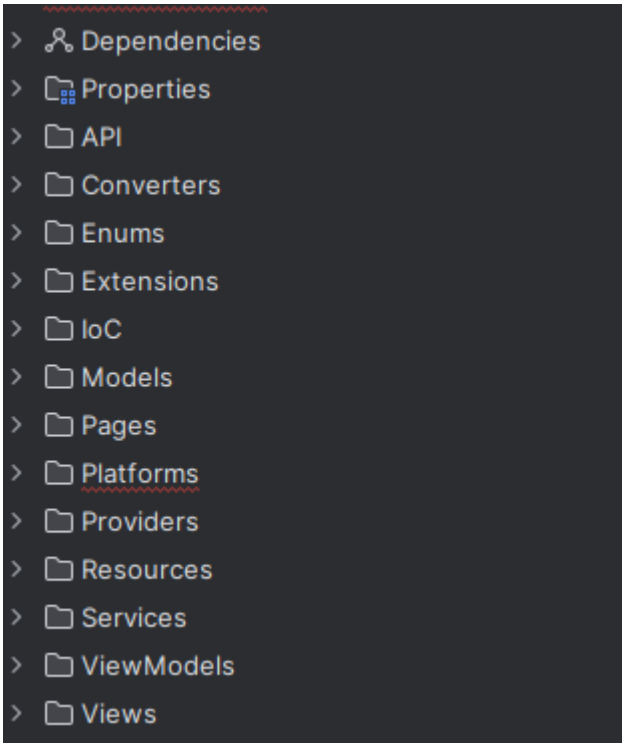

Рисунок 3.1 – Структура проекту

Dependencies включають зовнішні бібліотеки та пакети, які забезпечують функціональність застосунку, наприклад, бібліотеки для взаємодії з картографічними сервісами чи обробки запитів. Properties містять налаштування конфігурації проєкту, такі як AssemblyInfo.cs або інші параметри, що визначають загальні властивості застосунку. API складається з класів для роботи із запитами, необхідними для взаємодії з картографічними API або серверною частиною [6].

Converters використовуються для перетворення даних між View і ViewModel [3]. Наприклад, IValueConverter дозволяє змінювати формат або структуру даних під час їх відображення у View. Enums включають перерахування, які допомагають визначати статичні значення чи константи в застосунку. Extensions містять розширення методів для існуючих класів, що полегшує їх використання або додає нову функціональність.

IoC відповідає за реєстрацію сервісів і класів у контейнері інверсії управління, що дозволяє організувати залежності між компонентами. Models представляють класи моделі, які забезпечують зберігання даних і реалізують логіку доступу до них. Pages включають XAML-сторінки, що є основою UI та дозволяють реалізовувати навігацію у додатку.

Platforms містять платформоспецифічний код для підтримки Android, iOS, Windows або macOS. Providers надають класи-провайдери, які забезпечують доступ до даних або сервісів, таких як отримання даних з API або локальних баз даних. Resources включають статичні ресурси, такі як стилі, зображення, локалізовані рядки та інші елементи, що використовуються у застосунку.

Services забезпечують логіку обробки, наприклад, управління навігацією, збереженням даних або взаємодією з API. ViewModels реалізують патерн MVVM і забезпечують зв'язок між інтерфейсом користувача (UI) і даними, що надаються моделями. Views відповідають за відображення інтерфейсу користувача і можуть включати специфічні класи, які реалізують логіку відображення або взаємодії з користувачем [11].

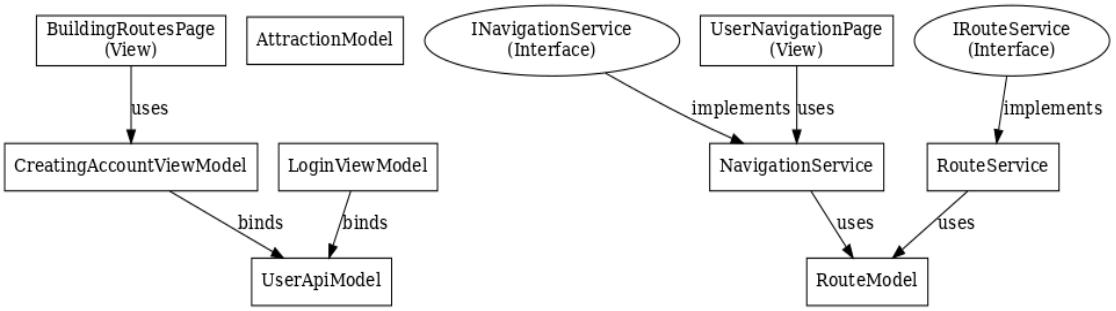

Рисунок 3.2 – Діаграма класів проекту

Загальний опис:

Проєкт побудовано за принципами MVVM. Views відповідають за

інтерфейс, ViewModels обробляють бізнес-логіку, а Models зберігають дані. API, Services та Providers відповідають за інтеграцію та обслуговування функцій додатку. IoC забезпечує впровадження залежностей, а Resources та Converters оптимізують вигляд і поведінку інтерфейсу.

3.2 Тестування бекенд сервісу

Для тестування бекенд сервісу використовуватимемо Postman:

Надсилаємо Get запит щоб отримати інформацію про всі найближчі місця від певної точки на мапі у радіусі 100 м. Приклад запиту: “http://localhost:3000/places/nearby?lat=49.870760&lon=24.056770&radius=100”

точка на карті згідно координат:

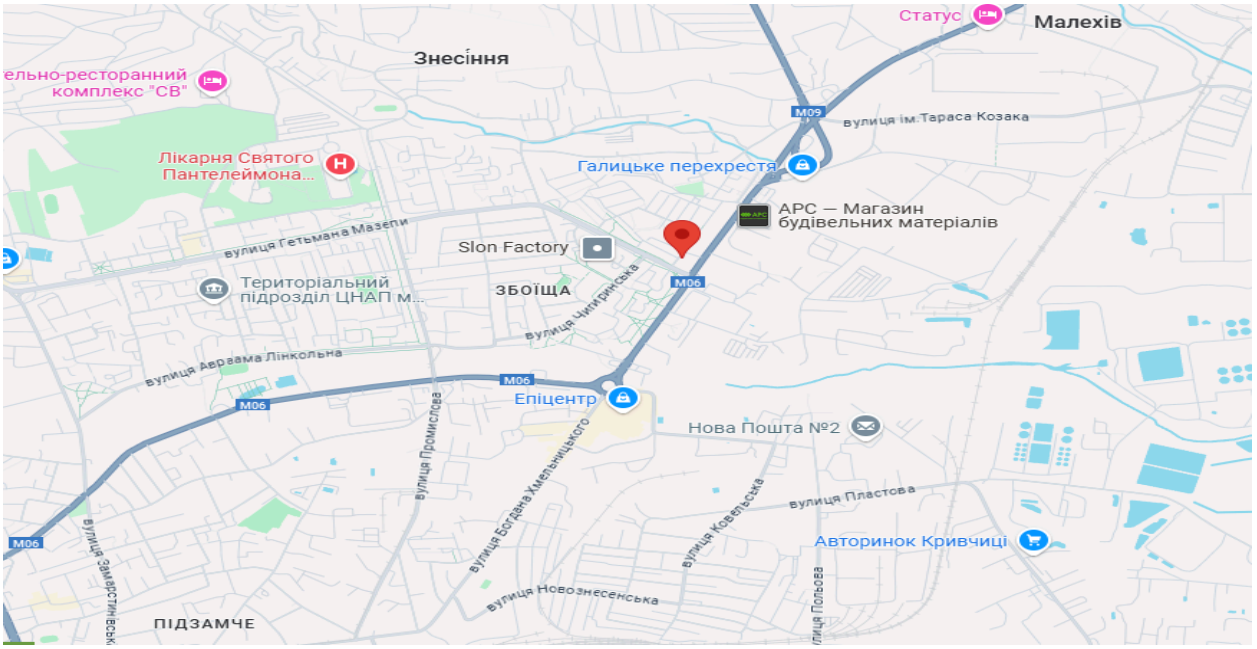

Рисунок 3.4 – Запит з координатами

Отримані результати:


```
{
    "fsq_id": "4fd73ca67716469f62f9f0c6",
    "categories": [
        {
            "id": 12075,
            "name": "Post Office",
            "short_name": "Post Office",
            "plural_name": "Post Offices",
            "icon": {
                "prefix": "https://ss3.4sqi.net/img/categories_v2/building/postoffice_",
                "suffix": ".png"
            }
        }
    ],
    "chains": [],
    "closed_bucket": "VeryLikelyOpen",
    "distance": 77,
    "geocodes": {
        "main": {
            "latitude": 49.870724,
            "longitude": 24.055683
        }
    },
    "link": "/v3/places/4fd73ca67716469f62f9f0c6",
    "location": {
        "address": "вул. Грінченка, 2а",
        "country": "UA",
        "formatted_address": "вул. Грінченка, 2а, Львів, 79059",
        "locality": "Львів",
        "postcode": "79059",
        "region": "Львівська обл."
    },
    "name": "Нова пошта №8",
    "related_places": {},
    "timezone": "Europe/Kiev"
}
```


Рисунок 3.5 – Отримані результати в радіусі 100м

Як можна помітити з рисунку, ,бекенд сервіс надає інформацю про актуальні місця і працює коректно відносно заданих параметрів.

Для перевірки функціоналу авторизації надішлемо ось такий POST запит:

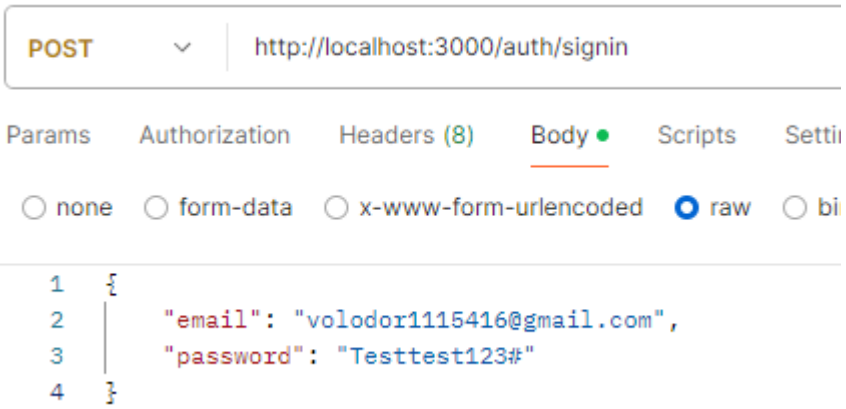

Рисунок 3.6 – Авторизація

У відповіді отримуємо токен, що використовується для авторизації в додатку та іншу інформацію по користувачу

```
"authenticationResult": {
    "idToken": "eyJraWQiOiJoOG9jOGoxWnF1bEhQdTd
        eyJzdWIiOiJjM2Q0MDhkMi1mMDIxLTcwOTItMjk
        0xX0NIYTFzb0pOMyIsImNvZ25pdG86dXNlcm5hb
        ZG4wN3M1dDkxOGw3NnNxY3Q4azIiLCJjdXN0b20
        I6ImlkIiwiYXV0aF90aW1lIjoxNzM0MjY1ODQyL
        bC5jb20ifQ.
        eGNpBm3pDqOHv9-YHLQNvm39N9nciyYlKKSmtlV
        VbdsHD-uWwuArp-3T3ISwa7IQ7IFA14U92lMGqf
    "accessToken": "eyJraWQiOiJCNlFzWVNrekRyMjh
        eyJzdWIiOiJjM2Q0MDhkMi1mMDIxLTcwOTItMjk
        QiOiIzbWhicjRkbjA3czV0OTE4bDc2c3FjdDhrM
        a2VuX3VzZSI6ImFjY2VzcyIsInNjb3BlIjoiYXd
        Q3N2YtYjQwNC01ZWIyMDM5NTdiMTgiLCJ1c2Vyb
        bWBZtsjPitE2fjdA8FI7lzFKjT9Jkt6ObPIlHlh
        UgsKzwuLmm6DSkxD2pVyItshgHFEJeqvGY2auIp
    "refreshToken": "eyJjdHkiOiJKV1QiLCJlbmMiOi
        rNTkKuro22e6z5mQVS1_AqhMqdaUZ8_b-4b6lFK
        GOpFsHt_AGUcqU6IyOd_6VM7Jrym7sL9DSxA1eq
        5B-LoVQhkkDNFXKTiGq3xfNQtbjc_ihe_Oa8gqF
        pyhYAU548PyGRVagssEJXZ0EyPkHdRQuaDi8sfD
        2TKKlAH4LEdycvRH83oKhRMUQwo3KWR2yxKBZgC
        Xh5w9qBM796fOVS8aGcClqQ_SoQ19vVS6l2vdcb
        fHFcAbypeRN9cJPTHrJahhbhTA78tofSBCXNTvB
        RirPpD9a6c9aPRc9Dm4pQsuaAbgG1D-yDtQK_jo
        wiO1XmZgHe08ypZGT0D5mOsXWP8zETmWav-sWUO
        ysVscfo45eQh2RE1Q09nz5wnCuc7paKnQsc7FYU
    "expiresIn": 3600,
    "tokenType": "Bearer"
}
```


Рисунок 3.7 – Отримані дані для авторизації з бекендом

Також є можливість перевірити чи користувач авторизований за допомогою GET запиту “http://localhost:3000/auth/me”

3.3.Тестування мобільного додатку:

Роботу мобільного додатку продемонстровано скріншотами з емулятора Apple Iphone 15 Max (iOS 17.5)

Авторизація користувача:

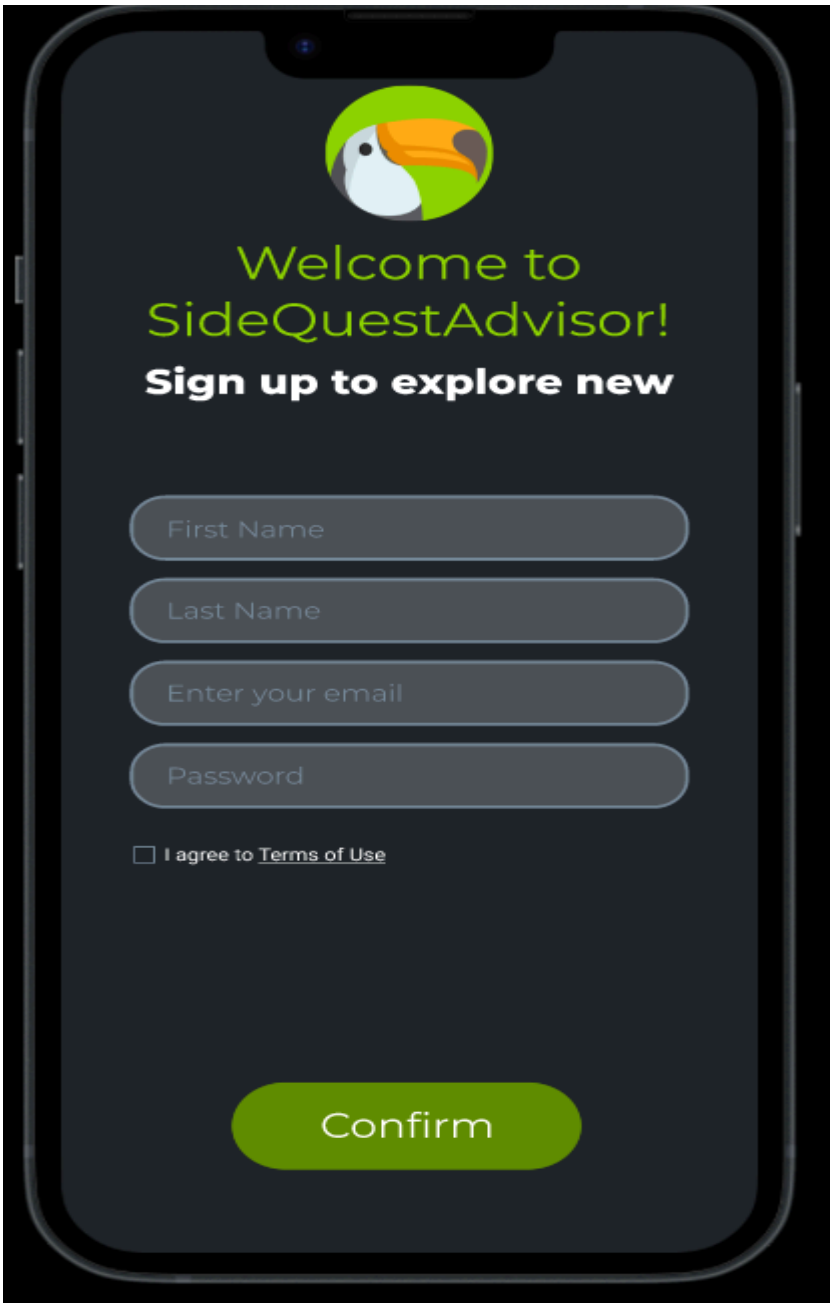

Рисунок 3.8 – Авторизація в мобільному додатку

Початковий вибір атракцій:

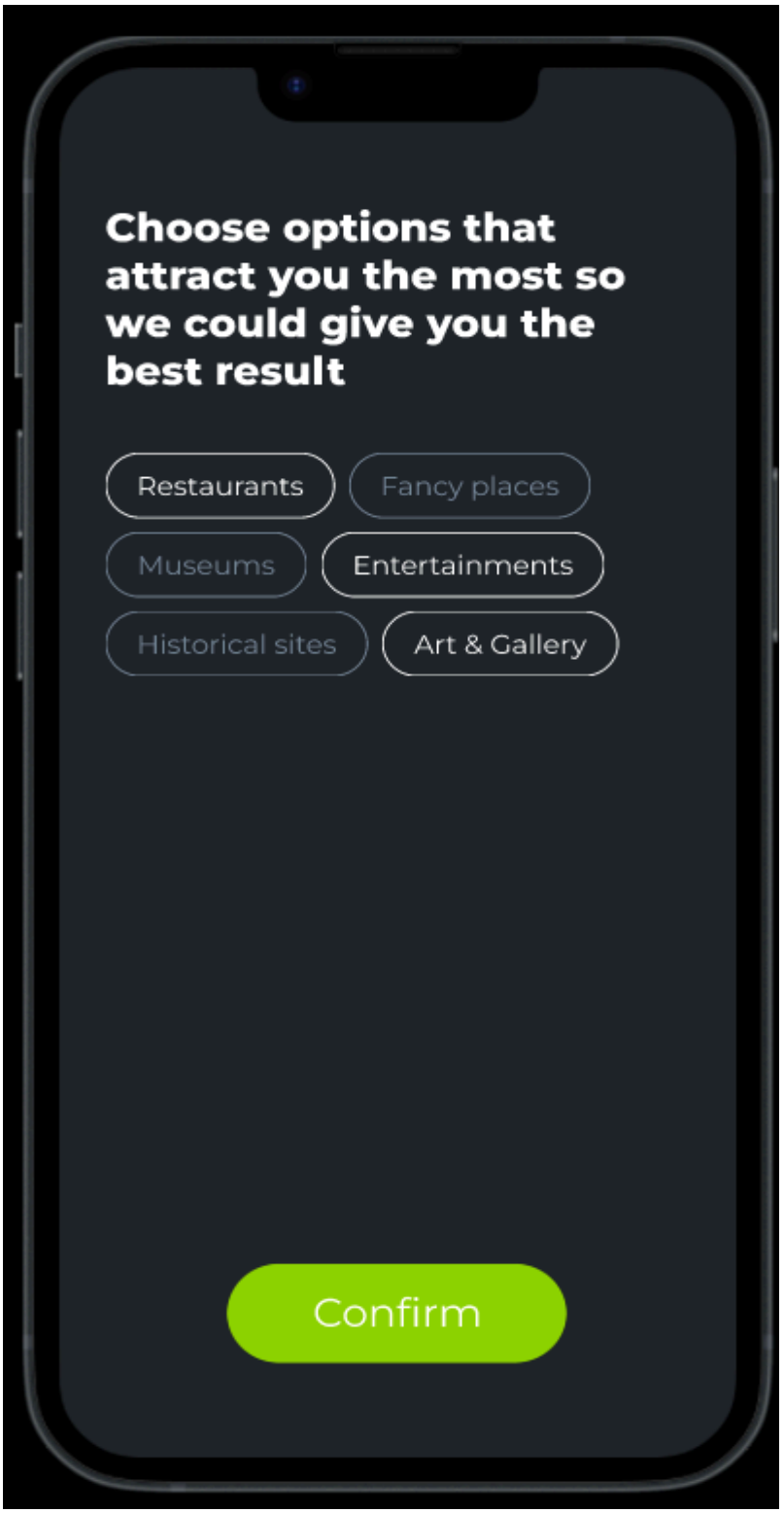

Рисунок 3.9 – Вибір типів улюблених атракцій

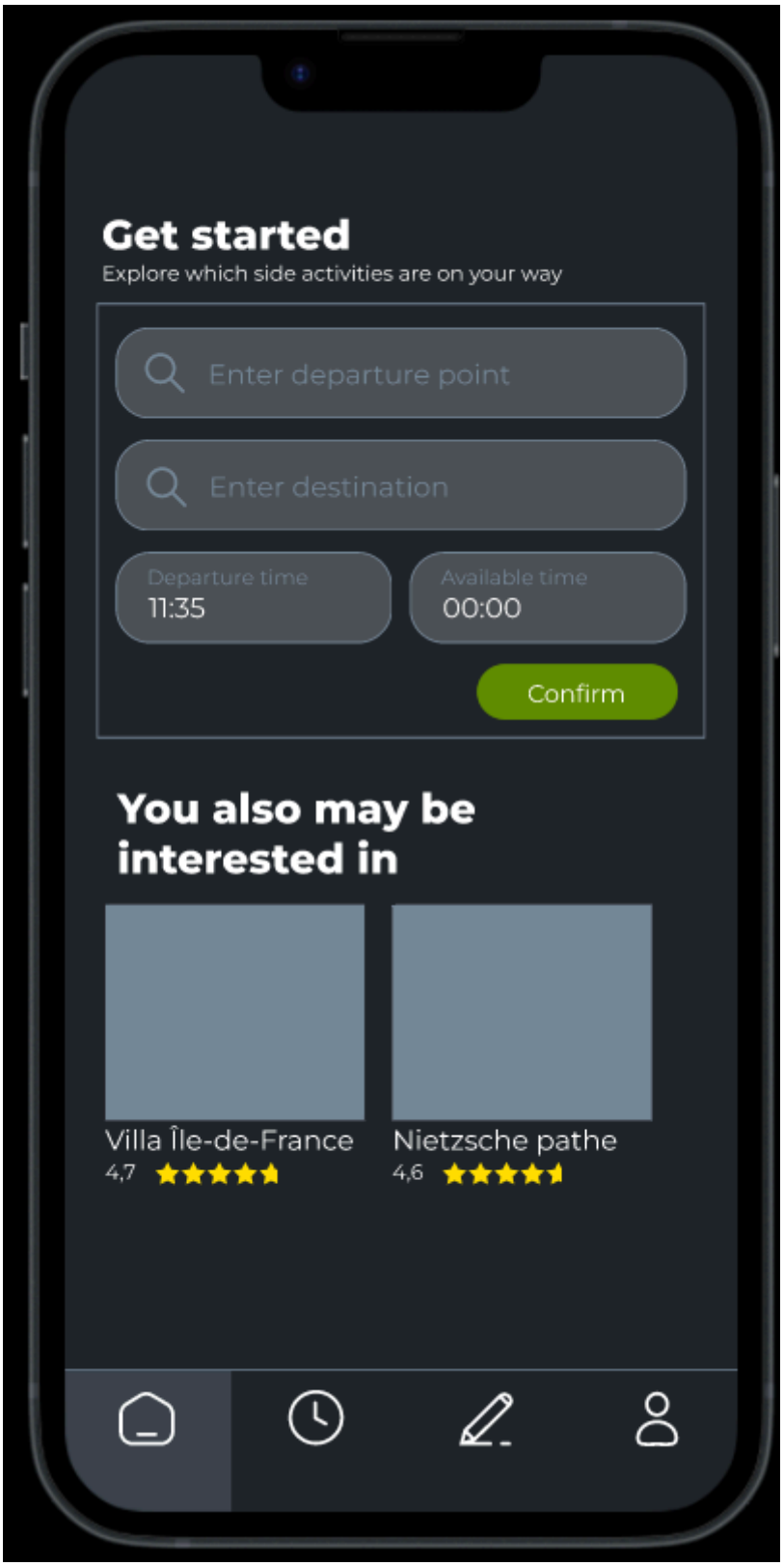

Рисунок 3.10 – Основне меню

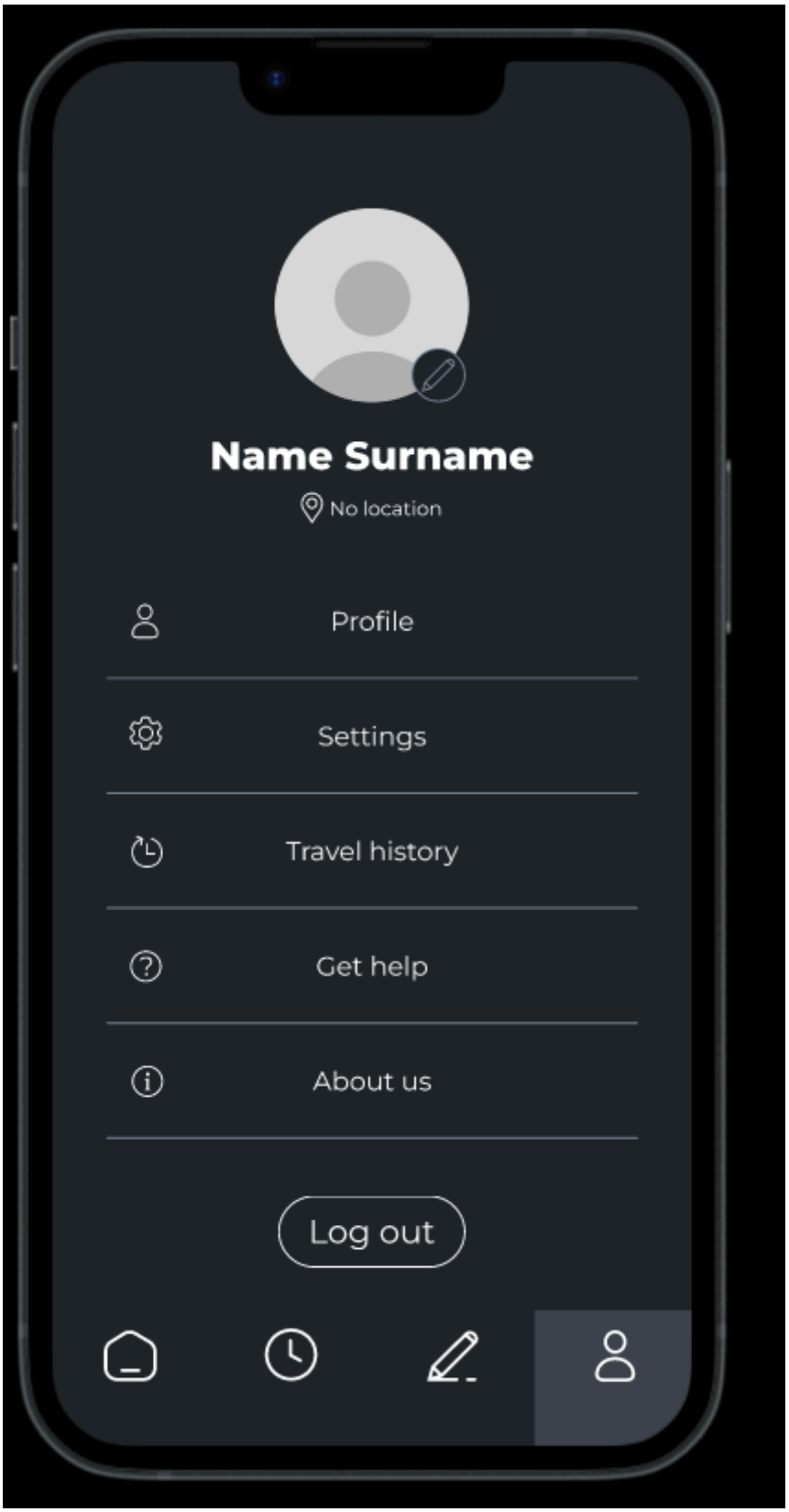

Рисунок 3.11 – Кабінет користувача

На рисунку 3.11 зображенно опції користуваяа для персоналізації свого акаунту

Вибір точок та побудова маршруту:

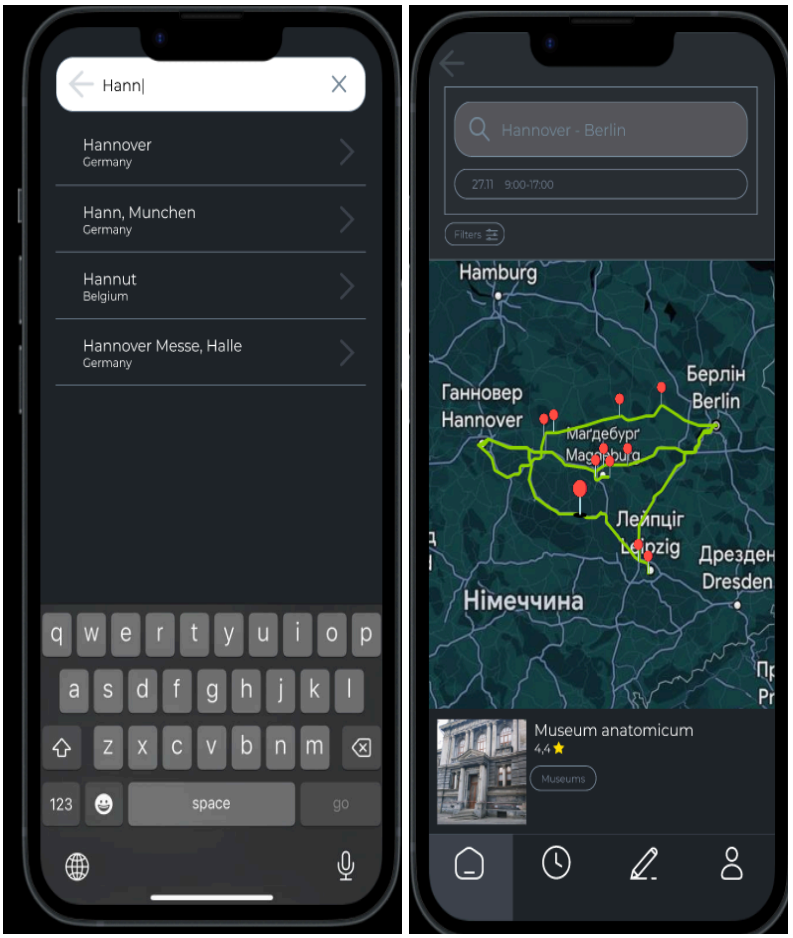

Рисунок 3.12 – Основний функціонал

На рисунку 3.12 зображено скріншоти з основним функціоналом додатку, вибір користувачем маршрутів, та пропозиції щодо подорожі, з можливістю оглянути цікаві для користувача місця, оскільки, додаток поки не є розміщений на популярних платформах, тому на рисунку з побудовою маршруту показано демонстраційний варінт атракції.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1 Охорона праці

При розробці навігаційного додатку з використанням .NET MAUI та картографічних API необхідно дотримуватися вимог з охорони праці, оскільки робота програмістів передбачає тривале перебування за комп’ютером у стаціонарних умовах, що може впливати на стан здоров’я. Для забезпечення безпечних і комфортних умов роботи варто врахувати основні положення «Правил охорони праці під час експлуатації електронно-обчислювальних машин» та «Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». Ці нормативні документи визначають вимоги до приміщень, робочих місць, освітлення, шуму, вібрації, мікроклімату та ергономіки.

Робочі місця програмістів мають бути розташовані в приміщеннях, які відповідають будівельним та санітарним нормам. Мінімальна площа приміщення повинна становити 6 квадратних метрів на одну людину, а об’єм – не менше 20 кубічних метрів. У приміщенні необхідно забезпечити оптимальний мікроклімат: температура повітря має бути в межах 18-22°C, відносна вологість – 40-60%, а швидкість руху повітря – не більше 0,2 м/с. Для підтримання цих параметрів слід використовувати системи опалення, кондиціонування або припливно-витяжну вентиляцію.

Приміщення повинні бути обладнані відповідно до правил ергономіки. Робочі столи мають бути оснащені регульованою робочою поверхнею з висотою в межах 680-800 мм, а їх ширина і глибина повинні забезпечувати достатній простір для розташування комп’ютера, клавіатури, документів та інших необхідних матеріалів. Робочі стільці повинні мати регульовану висоту та спинку, що підтримує поперековий відділ хребта. Розташування комп’ютера повинно забезпечувати відстань від очей користувача до екрана у межах 50-70 см.

Важливим фактором є освітлення приміщення. Природне освітлення має забезпечувати рівень освітленості, достатній для роботи з документами та комп'ютером, але при цьому не створювати відблисків на екрані. Для регулювання освітлення варто використовувати жалюзі або штори. Штучне освітлення повинно бути рівномірним і мати яскравість у межах нормативних значень. Слід уникати прямої близькості джерел світла до екранів дисплеїв, щоб уникнути відблисків. Поверхні стін, стелі та підлоги приміщення мають бути виконані з матеріалів, що мають матову поверхню для мінімізації відбиття світла.

Робочі місця повинні відповідати вимогам електробезпеки. Усі електроприлади, зокрема комп'ютери та периферійні пристрої, мають бути заземлені. Елементи систем опалення, водопостачання або інші заземлені конструкції повинні бути захищені діелектричними матеріалами для уникнення контакту працівників із потенційно небезпечними джерелами струму. Приміщення також мають бути обладнані системою пожежної сигналізації та вогнегасниками відповідно до чинних норм. Доступ до засобів пожежогасіння повинен бути завжди вільним.

У приміщенні необхідно забезпечити регулярне вологе прибирання для підтримання чистоти та зменшення накопичення пилу, який може негативно впливати на роботу електронного обладнання. Додатково, приміщення мають бути обладнані аптечками першої медичної допомоги. Для зменшення рівня втоми працівників варто організувати кімнати відпочинку або зони психологічного розвантаження, де співробітники зможуть зробити перерву під час роботи.

При розташуванні комп'ютерів важливо дотримуватися таких норм: відстань між бічними поверхнями комп'ютерів має бути не менше 1,2 метра, а відстань між тильною поверхнею одного монітора і екраном іншого – не менше 2,5 метра. У разі необхідності концентрації уваги операторів слід розділяти робочі місця перегородками висотою від 1,5 до 2 метрів.

Для захисту очей працівників можна використовувати локальні світлофільтри або сертифіковані приекранні фільтри. Також рекомендується

проводити регулярні перерви в роботі за комп’ютером для зменшення втоми очей та уникнення порушень зору.

Розробка навігаційного додатку з використанням .NET MAUI проводиться з урахуванням основних етапів життєвого циклу ПЗ. Це передбачає тривалу роботу над проектом, тому дотримання наведених вимог забезпечує не лише комфортні умови для розробників, але й безпеку їхньої праці. Усі ці аспекти є важливими як під час створення програмного забезпечення, так і під час його подальшого використання кінцевими споживачами.

4.2 Дослідження стійкості роботи систем обробки даних в умовах надзвичайних ситуацій.

Стійкість роботи систем обробки даних є ключовим чинником у забезпеченні їхньої безперебійної роботи під час надзвичайних ситуацій. Надзвичайні ситуації, викликані природними або техногенними факторами, можуть спричинити серйозні збої в роботі інформаційних систем, що, своєю чергою, здатне призвести до втрати даних, зупинки бізнес-процесів та інших негативних наслідків.

Згідно з посібником "Безпека в надзвичайних ситуаціях" (В.С. Стручок, 2022), забезпечення стійкості систем вимагає багаторівневого підходу, який включає як технічні, так і організаційні заходи. Одним із ключових елементів є резервування систем, що дозволяє зберігати дані та забезпечувати доступ до них навіть у разі аварій. Використання резервного копіювання, дублювання серверів і географічно розподілених дата-центрів є важливими аспектами забезпечення надійності роботи систем.

Ще одним важливим аспектом є використання автоматизованих систем моніторингу та раннього виявлення загроз. Як зазначено у навчальному посібнику "Техноекологія та цивільна безпека" (В.С. Стручок, 2022), впровадження таких систем дозволяє виявляти потенційно небезпечні ситуації на ранніх етапах і своєчасно реагувати на них. Це може включати як попередження про перевищення температури в серверній кімнаті, так і виявлення кібератак.

Для зниження впливу надзвичайних ситуацій на роботу систем також важливими є заходи з підготовки персоналу. Це включає тренування з моделювання різних сценаріїв НС, навчання алгоритмам дій у випадках аварій та забезпечення працівників необхідними засобами захисту. Впровадження планів відновлення після аварій (Disaster Recovery Plans) дозволяє мінімізувати час простою і швидко повернути системи до нормального стану.

Крім технічних рішень, дослідження показують, що важливою є організація ефективної комунікації між членами команди, що займається підтримкою роботи систем. У кризових ситуаціях чітке та оперативне обмінювання інформацією між відповідальними особами є критичним фактором для зменшення негативних наслідків.

Таким чином, дослідження стійкості роботи систем обробки даних в умовах надзвичайних ситуацій підтверджує важливість інтеграції технічних і організаційних заходів для забезпечення надійності їхньої роботи. Виконання рекомендацій, викладених у відповідних посібниках, дозволяє мінімізувати ризики та забезпечити безперервність функціонування інформаційних систем навіть у найскладніших умовах.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досліджено сучасні можливості платформи .NET MAUI для створення багатоплатформних додатків з використанням картографічних API. Основною метою дослідження було створення ефективного рішення для навігаційного додатку, здатного надавати користувачам функціонал подібний до популярних рішень, таких як Waze, із врахуванням сучасних вимог до продуктивності, зручності використання та інтеграції з хмарними сервісами.

Розроблений додаток продемонстрував практичність та ефективність використання .NET MAUI у контексті обробки даних картографічних API. Інтеграція картографічних сервісів, таких як Google Maps або OpenStreetMap, дозволила реалізувати функції відображення маршруту, побудови оптимального шляху, а також надання користувачам інтерактивного інтерфейсу для взаємодії з картою. Успішно впроваджена підтримка навігації та роботи з геолокацією значно підвищує функціональність додатку, відповідаючи потребам користувачів у сучасних навігаційних рішеннях.

Бекенд-сервіс, розроблений на основі Node.js та Amazon Web Services (AWS), продемонстрував високу масштабованість і надійність. Використання Single Sign-On (SSO) для аутентифікації забезпечило зручність для користувачів та покращило безпеку роботи з обліковими записами. Додатково, впровадження контейнеризації через Docker спростило процес тестування та розгортання сервісу, що є важливим аспектом для сучасних масштабованих додатків.

Під час виконання роботи було продемонстровано здатність .NET MAUI забезпечити високу продуктивність і сумісність з різними платформами, такими як Android, iOS та Windows. Використання єдиного коду дозволило зменшити час розробки та спростити підтримку додатку. Успішна інтеграція з хмарними сервісами і локальними API продемонструвала можливість побудови комплексних систем з використанням сучасних технологій.

Таким чином, виконання даної роботи підтвердило доцільність і

перспективність використання .NET MAUI у створенні багатоплатформних навігаційних додатків. Отримані результати можуть бути використані як основа для подальших досліджень та розробки додатків у сфері навігації, туризму або інших суміжних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft. "Introduction to .NET MAUI" [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/maui/.

2. SQLite. "What is SQLite?" [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/.

3. Krzysztof Cwalina, Jeremy Barton. "Pro .NET MAUI: Cross-Platform App Development". – Apress, 2023. – 450 с.

4. Microsoft Azure Documentation. "Azure Cloud Services Integration with .NET" [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure.

5. Boyko, I. , Mudryk, I. , Petryk, M. , Petryk, M. High-Performance Adsorption Modeling Methods with Feedback-Influynces in n-component Nanoporous Media Proceedings - International Conference on Advanced Computer Information Technologies, ACIT, 2021, pp. 441–444.

6. Гук, В. П. "Кросплатформний додаток на основі .NET: теорія та практика" // Наукові записки НУ ЛП, 2022. – №12. – С. 115-123.

7. Коваленко І. О., Іваненко Р. В. "Особливості використання SQLite для мобільних додатків" // Інформаційні технології та комп'ютерні науки. – 2021. – № 3. – С. 67-72.

8. OpenStreetMap. "Using OpenStreetMap API in Applications" [Електронний ресурс]. – Режим доступу: https://wiki.openstreetmap.org/.

9. Amazon Web Services. "AWS SDK for .NET Developer Guide" [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/sdk-for-net/.

10. David Ortinau. "Building Cross-Platform Applications with .NET MAUI" // .NET Blog, Microsoft. – 2022. – [Електронний ресурс]. – Режим доступу:

https://devblogs.microsoft.com/dotnet.

11. Petryk, M. , Mykhalyk, D. , Petryk, M. , Boyko, I. , Mudryk, I. Modeling of adsorption and desorption of hydrocarbons in nanoporous catalyst zeolite using nonlinear langmuir's isotherm. CEUR Workshop Proceedings, 2018, 2300, pp. 42–45

12. Halvorsen, Bjorn. "Database Management with SQLite in Mobile Applications" // Journal of Mobile Development, 2021. – № 5. – С. 42-55.

13. Waze Documentation. "Waze API and Real-Time Traffic Data" [Електронний ресурс]. – Режим доступу: https://developers.google.com/waze.

14. Here Technologies. "HERE Maps API for Developers" [Електронний ресурс]. – Режим доступу: https://developer.here.com/.

15. MobiDev. "Best Practices for Mobile Navigation Applications" // MobiDev Insights, 2023. – Режим доступу: https://mobidev.biz/blog.

ДОДАТКИ

Додаток А

Тези

УДК 004.9, 004.45
В. Глива, Ph.D. І. Мудрик
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ЕНТЕРПРАЙЗ ПАТЕРНИ ДЛЯ КРОСПЛАТФОРМНОЇ РОЗРОБКИ

V. Hlyva, Ph.D. I. Mudryk

ENTERPRISE PATTERNS FOR CROSS-PLATFORM DEVELOPMENT

У сучасній кросплатформній розробці програмних продуктів дедалі більшої популярності набувають ентерпрайз патерни, які забезпечують масштабованість, підтримку та ефективну інтеграцію. Основними викликами в цьому процесі є забезпечення продуктивності, спрощення підтримки та гарантування коректної роботи програмного забезпечення на різних платформах.

Дослідження демонструє ефективність використання таких патернів, як Repository, Unit of Work, Dependency Injection та Event Aggregator. Repository забезпечує ізоляцію бізнес-логіки від джерел даних, тоді як Unit of Work допомагає ефективно управляти транзакціями. Dependency Injection спрощує управління залежностями, що є критичним для великих кросплатформних проєктів. Event Aggregator сприяє зменшенню зв'язності між компонентами в багатоплатформних системах. Всі згадані патерни можуть використовуватись імплементації архітектурного стилю MVVM (Model-View-ViewModel). Суть MVVM полягає в максимально можливому розділенні обов'язків між сутностями системи, що приводить до можливості легко маштабувати та підтримувати кросплатформний застосунок.

Практичне впровадження цих патернів було протестоване у розробці додатків за допомогою .NET MAUI. Результати показали значне скорочення часу розробки та спрощення процесу масштабування системи.

Отримані результати підтверджують ефективність ентерпрайз патернів для побудови сучасних кросплатформних рішень і можуть бути використані як рекомендації для розробників програмного забезпечення.

Ключові слова: ентерпрайз патерни, кросплатформна розробка, .NET MAUI, Repository, Dependency Injection, масштабованість.

Література

1. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.
2. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.
3. Troelsen A., Japikse P. Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming. Apress, 2021.
4. Esposito D. Modern Web Development with ASP.NET Core 3: An end to end guide covering the latest features of Visual Studio 2019, .NET Core 3, and C# 8. Packt Publishing, 2019.

---

5. O Bryk, I Mudryk, M Holubovskyi, Y Stoianov. Machine learning models and methods aspects of processing unstructured data. Proceedings of the 1st International Workshop on Bioinformatics and Applied Information Technologies (BAIT 2024) Zboriv, Ukraine, 2024. pp. 64-74,

А. П. Бортняк, О. А. Пастух
АВТОМАТИЗАЦІЯ КАДРОВОЇ РОБОТИ ПІДПРИЄМСТВ
A. P. Bortnyak, O. A. Pastukh
AUTOMATION OF ENTERPRISE HR WORK 160

В. Герасим, Д. Михалик
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СОРТУВАЛЬНИХ ТА ОЧИЩУВАЛЬНИХ МАШИН
V. Herasym, D. Mykhalyk
SOFTWARE DEVELOPMENT FOR SORTING AND CLEANING MACHINES 161

Н. Б. Войцеховський, Ю. Б. Паляниця,
ОБҐРУНТУВАННЯ МЕТОДУ ПЕРЕДАЧІ ІНФОРМАЦІЙНОГО СИГНАЛУ ДЛЯ ПІДВИЩЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ СУПУТНИКОВИХ СИСТЕМ ЗВ'ЯЗКУ.
N. B. Voytsekhovskyi, Y. B. Palyanytsia
JUSTIFICATION OF THE METHOD OF INFORMATION SIGNAL TRANSMISSION TO INCREASE THE BANDWIDTH OF SATELLITE COMMUNICATION SYSTEMS 162

І. Гаврилюк
РОЗРОБКА ТА МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ФІНАНСОВОГО АУДИТУ ТА ОПТИМІЗАЦІЇ БІЗНЕС-ПРОЦЕСІВ КОМЕРЦІЙНИХ ПІДПРИЄМСТВ
I. Havryliuk
DEVELOPMENT AND MODELING OF INFORMATION SYSTEM FOR AUTOMATION OF FINANCE AUDIT AND OPTIMIZATION OF BUSINESS-PROCESSES OF COMMERCIAL ORGANIZATIONS 164

А. В. Гарасівка, А. М. Лупенко
ЗАЛЕЖНІСТЬ ШВИДКОСТІ ПЕРЕДАЧІ ІНФОРМАЦІЇ ВІД ГЕОГРАФІЧНОГО РОЗТАШУВАННЯ СЕРВЕРІВ ХМАРНИХ СХОВИЩ
A. V. Harasivka, A. M. Lupenko
DEPENDENCE OF INFORMATION TRANSFER SPEED ON THE GEOGRAPHICAL LOCATION OF REMOTE CLOUD STORAGE SERVERS 165

В. Глива, І. Мудрик
ЕНТЕРПРАЙЗ ПАТЕРНИ ДЛЯ КРОСПЛАТФОРМНОЇ РОЗРОБКИ
V. Hlyva, I. Mudryk
ENTERPRISE PATTERNS FOR CROSS-PLATFORM DEVELOPMENT 166

В. В. Деркач, Г. Б. Цуприк
РОЗРОБКА УНІВЕРСАЛЬНОЇ МОБІЛЬНОЇ WEB-СИСТЕМИ АВТОМАТИЗАЦІЇ ОБЛІКУ РЕСУРСІВ БІЗНЕС-СТРУКТУРИ
V.V. Derkach, H. B. Tsupryk
DEVELOPMENT OF A UNIVERSAL MOBILE WEB-SYSTEM FOR AUTOMATION OF BUSINESS-STRUCTURE RESOURCES 167

М. Дмитраш
РОЗГОРТАННЯ РЕСУРС, ЯКИЙ БУЛО РОЗРОБЛЕНО З ВИКОРИСТАННЯМ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ
M. Dmytrash
DEPLOYMENT OF RESOURCE THAT WAS DEVELOPED USING MICROSERVICE ARCHITECTURE 169

О. А. Заяць
С4-ДІАГРАМИ ЯК ГРАФІЧНИЙ МЕТОД ОПИСУ ВИМОГ І АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
O. A. Zaiats
C4-DIAGRAMS AS A GRAPHICAL METHOD FOR DESCRIBING SOFTWARE REQUIREMENTS AND ARCHITECTURE 170

Додаток Б

Код основних програмних модулів:

NavigationService.cs

```
using System.Globalization;
using SideQuestAdvisor.IoC.Providers;
using SideQuestAdvisor.ViewModels.Base;

namespace SideQuestAdvisor.Services;

public class NavigationService : INavigationService
{
   private const string ViewModelString = "ViewModel";
   private const string PageString = "Page";

   private readonly INavigation navigation = GetNavigation();

   public async Task NavigateAsync<TViewModel>(object? parameter =
null) where TViewModel : BaseViewModel
   {
       await
InternalNavigateToAsync<TViewModel>(typeof(TViewModel),
parameter!);
   }

   public async Task CloseAsync()
   {
       await navigation.PopAsync();
   }

   private async Task InternalNavigateToAsync<TViewModel>(Type
viewModelType, object parameter) where TViewModel : BaseViewModel
   {
       var page = CreatePage(viewModelType, parameter);

       var viewModel = GetViewModel<TViewModel>();

       await viewModel.Init(parameter);

       BindViewModelToPage(page, viewModel);

       await navigation.PushAsync(page);
   }

   private Type GetPageTypeForViewModel(Type viewModelType)
   {
       var pagesNamespace =
viewModelType.Namespace?.Replace(ViewModelString, PageString);
       var pageClassName =
viewModelType.Name.Replace(ViewModelString, PageString);
       var assemblyName = typeof(App).Assembly.FullName;

       return FindType(pagesNamespace!, pageClassName,
assemblyName!, string.Empty);
   }

   private Page CreatePage(Type viewModelType, object parameter)
```

```
    {
        var pageType = GetPageTypeForViewModel(viewModelType);
        if (pageType == null)
        {
            throw new Exception($"Cannot locate page type for
{viewModelType}");
        }

        var page = Activator.CreateInstance(pageType) as Page;

        return page!;
    }

    private void BindViewModelToPage(Page targetPage, BaseViewModel
targetViewModel)
    {
        targetPage.BindingContext = targetViewModel;
        targetPage.Appearing += OnPageAppearing;
        targetPage.Disappearing += OnPageDisappearing;
    }

    private async void OnPageDisappearing(object? sender, EventArgs
e)
    {
        if (sender is Page { BindingContext: BaseViewModel
baseViewModel })
        {
            await baseViewModel.OnDisappearing();
        }
    }

    private async void OnPageAppearing(object? sender, EventArgs e)
    {
        if (sender is Page { BindingContext: BaseViewModel
baseViewModel })
        {
            await baseViewModel.OnAppearing();
        }
    }

    private static Type FindType(string pageNameSpace, string
pageClassName, string assemblyName,
        params string[] includedFolders)
    {
        foreach (var folder in includedFolders)
        {
            string fullTypeName;

            if (string.IsNullOrWhiteSpace(folder))
            {
                fullTypeName =
string.Format(CultureInfo.CurrentCulture, "{0}.{1}, {2}",
pageNameSpace,
                    pageClassName, assemblyName);
            }
            else
            {
                fullTypeName =
string.Format(CultureInfo.CurrentCulture, "{0}.{1}.{2}, {3}",
```

```
pageNameSpace, folder,
                    pageClassName, assemblyName);
            }

            var type = Type.GetType(fullTypeName);

            if (type != null)
            {
                return type;
            }
        }

        var error = string.Format(CultureInfo.CurrentCulture,
$"Can't find page {pageClassName}");
        Console.Write(error);

        throw new ArgumentException(error);
    }

    private static INavigation GetNavigation()
    {
        var navigationPage = new NavigationPage();

        Application.Current!.MainPage = navigationPage;

        return navigationPage.Navigation;
    }

    private static T GetViewModel<T>()
    {
        return ServiceHelper.GetService<T>()!;
    }
}
```

EnterTripParametersViewModel.cs

```
using System.Windows.Input;
using CommunityToolkit.Mvvm.Input;
using SideQuestAdvisor.Models;
using SideQuestAdvisor.Providers;
using SideQuestAdvisor.Services;
using SideQuestAdvisor.ViewModels.Base;
using SideQuestAdvisor.ViewModels.Map;

namespace SideQuestAdvisor.ViewModels;

public class EnterTripParametersViewModel : BaseViewModel
{
    private readonly INavigationService _navigationService;
    private readonly IApiDataProvider _apiDataProvider;

    private UserModel? _currentUserModel;
    public string DeparturePoint { get; set; } = "Enter departure
point";
    public string DestinationPoint { get; set; } = "Enter
destination point";
    public TimeSpan StartTime { get; set; } =
TimeSpan.FromHours(DateTime.Now.Hour);
    public TimeSpan EndTime { get; set; } =
TimeSpan.FromHours(DateTime.Now.Hour);
```

```
   public ICommand BuildARouteCommand { get; set; }

   public EnterTripParametersViewModel(INavigationService
navigationService, IApiDataProvider apiDataProvider) :
base(navigationService)
   {
       _navigationService = navigationService;
       _apiDataProvider = apiDataProvider;
       BuildARouteCommand = new
AsyncRelayCommand(BuildARouteCommandExecute);
   }

   internal override async Task Init(object parameter)
   {
       await base.Init(parameter);

       if (parameter is UserModel userModel)
       {
           _currentUserModel = userModel;
       }
   }

   private async Task BuildARouteCommandExecute()
   {
       var routeModel = new RouteModel
       {
           DeparturePoint = DeparturePoint,
           DestinationPoint = DestinationPoint,
           AvailableTime = GetAvailableTime()
       };

       await
_navigationService.NavigateAsync<BuildingRoutesViewModel>(routeMod
el);
   }

   private TimeSpan GetAvailableTime()
   {
       return TimeSpan.FromMinutes(EndTime.TotalMinutes -
StartTime.TotalMinutes);
   }
}
```

LoginViewModel.cs

```
using System.Net;
using System.Windows.Input;
using CommunityToolkit.Mvvm.Input;
using Newtonsoft.Json;
using PropertyChanged;
using SideQuestAdvisor.Models;
using SideQuestAdvisor.Models.Auth;
using SideQuestAdvisor.Providers;
using SideQuestAdvisor.Providers.Enums;
using SideQuestAdvisor.Services;
using SideQuestAdvisor.ViewModels.Base;

namespace SideQuestAdvisor.ViewModels;
[AddINotifyPropertyChangedInterface]
public class LoginViewModel : BaseViewModel
{
```

```
   private readonly INavigationService _navigationService;
   private readonly IApiDataProvider _apiDataProvider;
   public ICommand LoginCommand { get; set; }

   public ICommand SignUpCommand { get; set; }

   public UserModel UserModel { get; set; }

   public LoginViewModel(INavigationService navigationService,
IApiDataProvider apiDataProvider) : base(navigationService)
   {
       UserModel = new UserModel();

       UserModel.Password = "1234Te@st";

       UserModel.Email = "glivavasa@gmail.com";

       SignUpCommand = new
AsyncRelayCommand(SignUpCommandExecute);
       this._navigationService = navigationService;
       _apiDataProvider = apiDataProvider;
       LoginCommand = new
AsyncRelayCommand(LoginCommandExecuteAsync);
   }

   private async Task LoginCommandExecuteAsync()
   {
      var data = await
_apiDataProvider.GetDataResponseAsync<AuthData>(ActionType.SignIn,
UserModel);

      await
_navigationService.NavigateAsync<EnterTripParametersViewModel>(Use
rModel);
   }

   private async Task SignUpCommandExecute()
   {
       await _navigationService.NavigateAsync<SignUpViewModel>();
   }
}
```

LoginPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:converters="clr-namespace:SideQuestAdvisor.Converters"

xmlns:viewModels="clr-namespace:SideQuestAdvisor.ViewModels"
            x:Class="SideQuestAdvisor.Pages.LoginPage"
            Background="{StaticResource
PrimaryPageBackgroundColor}">
   <ContentPage.Content>
       <Grid x:DataType="viewModels:LoginViewModel">
           <Grid.RowDefinitions>
               <RowDefinition Height="Auto" />
               <RowDefinition Height="Auto" />
               <RowDefinition Height="Auto" />
```

```
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <Image
                HeightRequest="60"
                WidthRequest="60"
                Grid.Row="0"
                HorizontalOptions="Center"
                Margin="0,12,0,0"
                Source="pelican.svg" />
            <Label
                FontAttributes="Italic"
                FontSize="32"
                Grid.Row="1"
                HorizontalOptions="Center"
                HorizontalTextAlignment="Center"
                Text="Welcome to SideQuestAdvisor!"
                TextColor="{StaticResource
PrimaryButtonBackgroundColor}" />
            <Label
                FontAttributes="Bold"
                FontSize="24"
                Grid.Row="2"
                Margin="28,0,28,16"
                HorizontalOptions="Center"
                HorizontalTextAlignment="Center"
                Text="Log in or Sign up to make your route better"
                TextColor="{StaticResource
PrimarySelectedItemBackground}" />
            <VerticalStackLayout Grid.Row="3" Spacing="12"
HeightRequest="102">
                <VerticalStackLayout.Resources>
                    <converters:EntryTextToBackgroundColorConverter
                        EmptyColor="{StaticResource
PrimaryPageBackgroundColor}"
                        NotEmptyColor="White"
                        x:Key="EntryTextToBackgroundColorConverter"
/>
                </VerticalStackLayout.Resources>
                <Border
                    BackgroundColor="{Binding Source={x:Reference
FirstNameEntry}, Path=Text, Converter={StaticResource
EntryTextToBackgroundColorConverter}}"
                    Style="{StaticResource LoginPageBorderStyle}">
                    <Border.StrokeShape>
                        <RoundRectangle CornerRadius="20" />
                    </Border.StrokeShape>
                    <Entry
                        Style="{StaticResource LoginEntryStyle}"
                        Placeholder="Enter your email"
                        Text="{Binding UserModel.Email}"
                        x:Name="FirstNameEntry" />
                </Border>
                <Border
                    BackgroundColor="{Binding Source={x:Reference
```

```
PasswordEntry}, Path=Text, Converter={StaticResource
EntryTextToBackgroundColorConverter}}"
                    Style="{StaticResource LoginPageBorderStyle}">
                    <Border.StrokeShape>
                        <RoundRectangle CornerRadius="20" />
                    </Border.StrokeShape>
                    <Entry
                        IsPassword="True"
                        Style="{StaticResource LoginEntryStyle}"
                        Placeholder="Password"
                        Text="{Binding UserModel.Password}"
                        x:Name="PasswordEntry" />
                </Border>
            </VerticalStackLayout>
            <Button Grid.Row="4" VerticalOptions="Center"
BorderWidth="2"
                    Margin="0,12,0,0"
                    BorderColor="{StaticResource
PrimaryControlForegroundColor}"
                    BackgroundColor="{StaticResource
PrimaryPageBackgroundColor}"
                    Text="Log in"
                    FontSize="16"
                    Command="{Binding LoginCommand}"
                    FontFamily="Montserrat"
                    TextColor="{StaticResource
PrimaryControlForegroundColor}"
                    HeightRequest="50" WidthRequest="132"
CornerRadius="35">
            </Button>
            <Grid Grid.Row="5" Margin="0,16">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <Border Grid.Column="0" WidthRequest="130"
HeightRequest="2"
                        BackgroundColor="{StaticResource
PrimaryControlForegroundColor}" />
                <Label Grid.Column="1" Text="OR"
Style="{StaticResource PrimaryTextStyle}"
                       TextColor="{StaticResource
PrimaryControlForegroundColor}"/>
                <Border Grid.Column="2" WidthRequest="130"
HeightRequest="2"
                        BackgroundColor="{StaticResource
PrimaryControlForegroundColor}"/>
            </Grid>
            <VerticalStackLayout Grid.Row="6" HeightRequest="117"
Spacing="12">
                <Button HeightRequest="50" WidthRequest="312"
ImageSource="google.png"
                        Text="Continue with Google"
                        BorderWidth="2"
                        BorderColor="{StaticResource
PrimaryControlForegroundColor}"
                        BackgroundColor="{StaticResource
PrimaryPageBackgroundColor}"
```

```
                        FontSize="16"
                        CornerRadius="20"
                        FontFamily="Montserrat"
                        ContentLayout="Left, 20"
                        TextColor="{StaticResource
PrimaryControlForegroundColor}">
                </Button>
                <Button HeightRequest="50" WidthRequest="312"
                        ImageSource="apple.png"
                        Text="Continue with Apple"
                        BorderWidth="2"
                        BorderColor="{StaticResource
PrimaryControlForegroundColor}"
                        BackgroundColor="{StaticResource
PrimaryPageBackgroundColor}"
                        FontSize="16"
                        CornerRadius="20"
                        FontFamily="Montserrat"
                        ContentLayout="Left, 20"
                        TextColor="{StaticResource
PrimaryControlForegroundColor}">
                </Button>
            </VerticalStackLayout>
            <Grid Grid.Row="7" Margin="0,16">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <Border Grid.Column="0" WidthRequest="57"
HeightRequest="2"
                        BackgroundColor="{StaticResource
PrimaryControlForegroundColor}" />
                <Label Grid.Column="1" Text="Don`t have account?"
Style="{StaticResource PrimaryTextStyle}"
                       TextColor="{StaticResource
PrimaryControlForegroundColor}"/>
                <Border Grid.Column="2" WidthRequest="57"
HeightRequest="2"
                        BackgroundColor="{StaticResource
PrimaryControlForegroundColor}"/>
            </Grid>
            <Button Grid.Row="8"
                    VerticalOptions="Center"
                    BorderWidth="2"
                    Margin="0,0,0,20"
                    BorderColor="{StaticResource
PrimaryControlForegroundColor}"
                    BackgroundColor="{StaticResource
PrimaryPageBackgroundColor}"
                    Text="Sign Up"
                    FontSize="16"
                    FontFamily="Montserrat"
                    Command="{Binding SignUpCommand}"
                    TextColor="{StaticResource
PrimaryControlForegroundColor}"
                    HeightRequest="50" WidthRequest="132"
CornerRadius="35">
            </Button>
```

```
        </Grid>
    </ContentPage.Content>
</ContentPage>
<?xml version="1.0" encoding="utf-8"?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SideQuestAdvisor.Pages.CreatingAccountPage"
             BackgroundColor="{StaticResource
PrimaryPageBackgroundColor}">
    <ContentPage.Content>
        <Grid VerticalOptions="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <Label Grid.Row="0"
                   TextColor="{StaticResource PrimaryTextColor}"
                   Text="Creating your account"
                   WidthRequest="295"
                   VerticalOptions="Start"
                   HeightRequest="98"
                   FontSize="40"
                   HorizontalTextAlignment="Center"
                   FontAttributes="Bold"
                   HorizontalOptions="Center" />
            <Image Grid.Row="1" Source="collectinginfoasgif.gif"
IsAnimationPlaying="True" HorizontalOptions="Center" />
        </Grid>
    </ContentPage.Content>
</ContentPage>
<?xml version="1.0" encoding="utf-8"?>
```

SingUpPage.xaml

```
<ContentPage
    Background="{StaticResource PrimaryPageBackgroundColor}"
    x:Class="SideQuestAdvisor.Pages.SignUpPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:converters="clr-namespace:SideQuestAdvisor.Converters"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewModels="clr-namespace:SideQuestAdvisor.ViewModels"
    x:DataType="viewModels:SignUpViewModel">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Image
            Grid.Row="0"
            HorizontalOptions="Center"
            Margin="0,36,0,8"
            Source="pelican.svg" />
        <Label
            FontAttributes="Italic"
            FontSize="32"
```

```
            Grid.Row="1"
            HorizontalOptions="Center"
            HorizontalTextAlignment="Center"
            Text="Welcome to SideQuestAdvisor!"
            TextColor="{StaticResource
PrimaryButtonBackgroundColor}" />
        <Label
            FontAttributes="Bold"
            FontSize="24"
            Grid.Row="2"
            Margin="0,8,0,64"
            HorizontalOptions="Center"
            HorizontalTextAlignment="Center"
            Text="Sign up to explore new"
            TextColor="{StaticResource
PrimarySelectedItemBackground}" />
        <StackLayout Grid.Row="3" Orientation="Vertical"
Spacing="12">
            <StackLayout.Resources>
                <converters:EntryTextToBackgroundColorConverter
                    EmptyColor="{StaticResource
PrimaryPageBackgroundColor}"
                    NotEmptyColor="White"
                    x:Key="EntryTextToBackgroundColorConverter" />
            </StackLayout.Resources>
            <Border
                BackgroundColor="{Binding Source={x:Reference
FirstNameEntry}, Path=Text, Converter={StaticResource
EntryTextToBackgroundColorConverter}}"
                Style="{StaticResource LoginPageBorderStyle}">
                <Border.StrokeShape>
                    <RoundRectangle CornerRadius="20" />
                </Border.StrokeShape>
                <Entry
                    Style="{StaticResource LoginEntryStyle}"
                    Placeholder="First name"
                    Text="{Binding SignUpUserModel.FirstName}"
                    x:Name="FirstNameEntry"/>
            </Border>
            <Border
                BackgroundColor="{Binding Source={x:Reference
LastNameEntry}, Path=Text, Converter={StaticResource
EntryTextToBackgroundColorConverter}}"
                Style="{StaticResource LoginPageBorderStyle}">
                <Border.StrokeShape>
                    <RoundRectangle CornerRadius="20" />
                </Border.StrokeShape>
                <Entry
                    Style="{StaticResource LoginEntryStyle}"
                    Placeholder="Last name"
                    Text="{Binding SignUpUserModel.LastName}"
                    x:Name="LastNameEntry" />
            </Border>
            <Border
                BackgroundColor="{Binding Source={x:Reference
EmailEntry}, Path=Text, Converter={StaticResource
EntryTextToBackgroundColorConverter}}"
                Style="{StaticResource LoginPageBorderStyle}">
                <Border.StrokeShape>
```

```
                    <RoundRectangle CornerRadius="20" />
                </Border.StrokeShape>
                <Entry
                    Style="{StaticResource LoginEntryStyle}"
                    Keyboard="Email"
                    Placeholder="Enter your email"
                    Text="{Binding SignUpUserModel.Email}"
                    x:Name="EmailEntry" />
            </Border>
            <Border
                BackgroundColor="{Binding Source={x:Reference
PasswordEntry}, Path=Text, Converter={StaticResource
EntryTextToBackgroundColorConverter}}"
                Style="{StaticResource LoginPageBorderStyle}">
                <Border.StrokeShape>
                    <RoundRectangle CornerRadius="20" />
                </Border.StrokeShape>
                <Entry
                    IsPassword="True"
                    Style="{StaticResource LoginEntryStyle}"
                    Placeholder="Password"
                    Text="{Binding SignUpUserModel.Password}"
                    x:Name="PasswordEntry" />
            </Border>
            <HorizontalStackLayout Margin="42,0,0,0">
                <CheckBox IsChecked="{Binding IsAgreedToTermsOfUse,
Mode=TwoWay}"
                          Color="{StaticResource
PrimaryControlForegroundColor}" />
                <Label VerticalOptions="Center" Text="I agree to
Terms of Use" FontSize="12" TextColor="White" />
            </HorizontalStackLayout>
        </StackLayout>
        <Button
            Grid.Row="4"
            Text="Confirm"
            Command="{Binding SignUpCommand}"
            Style="{StaticResource PrimaryButtonStyle}"/>
    </Grid>
</ContentPage>
```

TimeSpanPicker.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SideQuestAdvisor.Views;

public partial class TimeSpanPicker : ContentView
{
    public static readonly BindableProperty TimeSpanProperty =
        BindableProperty.Create(nameof(TimeSpan), typeof(TimeSpan),
typeof(TimeSpanPicker), TimeSpan.Zero,
            BindingMode.TwoWay, propertyChanged:
```

```
OnTimeSpanChanged);

    public TimeSpan TimeSpan
    {
        get => (TimeSpan)GetValue(TimeSpanProperty);
        set => SetValue(TimeSpanProperty, value);
    }

    public ObservableCollection<string> Hours { get; } = [];
    public ObservableCollection<string> Minutes { get; } = [];

    public TimeSpanPicker()
    {
        InitializeComponent();
        PopulatePickers();
        UpdatePickers();

        TimeSpan = new TimeSpan();
    }

    private void PopulatePickers()
    {
        for (var i = 0; i <= 23; i++)
        {
            Hours.Add(i.ToString());
        }

        for (var i = 0; i < 60; i += 1)
        {
            Minutes.Add(i.ToString());
        }
    }

    private static void OnTimeSpanChanged(BindableObject bindable,
object oldValue, object newValue)
    {
        var picker = (TimeSpanPicker)bindable;
        picker.UpdatePickers();
    }

    private void UpdatePickers()
    {
        HoursPicker.SelectedItem = TimeSpan.Hours;
        MinutesPicker.SelectedItem = TimeSpan.Minutes;

        var hours = TimeSpan.Hours.ToString();
        var minutes = TimeSpan.Minutes.ToString();

        if (hours.Length == 1)
        {
            HoursPicker.SelectedItem = $"0{hours}";
        }
        if (minutes.Length == 1)
        {
            MinutesPicker.SelectedItem = $"0{minutes}";
        }

        UpdatePickerTitles();
    }
```

```
    private void UpdatePickerTitles()
    {
        var hours = TimeSpan.Hours.ToString();
        var minutes = TimeSpan.Minutes.ToString();

        HoursPicker.Title = hours;
        MinutesPicker.Title = minutes;

        if (hours.Length == 1)
        {
            HoursPicker.Title = $"0{hours}";
        }
        if (minutes.Length == 1)
        {
            MinutesPicker.Title = $"0{minutes}";
        }
    }

    private void OnPickerSelectedIndexChanged(object sender,
EventArgs e)
    {
        if (HoursPicker.SelectedItem is string selectedHours)
        {
            TimeSpan = new TimeSpan(Convert.ToInt32(selectedHours),
TimeSpan.Minutes, 0);
            UpdatePickerTitles();
        }
        else if (MinutesPicker.SelectedItem is string
selectedMinutes)
        {
            TimeSpan = new TimeSpan(TimeSpan.Hours,
Convert.ToInt32(selectedMinutes), 0);
            UpdatePickerTitles();
        }
    }
}
```

Додаток В
Диск з роботою